Automated Detection of Non-Termination and `NullPointerExceptions` for Java Bytecode

M. Brockschmidt, T. Ströder, C. Otto, J. Giesl

LuFG Informatik 2, RWTH Aachen University, Germany

FoVeOOS 2011, Turin

Automated Non-Termination Analysis

- Logic programs:
  De Schreye '90 ..., Payet & Mesnard '06, ...
- TRSs and SRSs:
  Giesl et. al '05, Payet '06, ...
- C:
  Gupta et. al '08, ...
- JBC:
  Velroyen '08, Payet & Spoto '09

The example

```
class Loop {
 void main(String[] a){
  int i = 0;
  int j = a.length;
  while (i < j) {
   i += a[i].length();}}}
```

1. Adds up lengths.

The example

```
class Loop {
 void main(String[] a){
  int i = 0;
  int j = a.length;
  while (i < j) {
   i += a[i].length();}}}
```

1. Adds up lengths.
2. May not terminate.

The example

```
class Loop {
 void main(String[] a){
  int i = 0;
  int j = a.length;
  while (i < j) {
   i += a[i].length();}}}
```

1. Adds up lengths.
2. May not terminate.
3. May throw `NullPointerExc`

The example

```
main(String[] a):
00: iconst_0        #load 0 to stack
01: istore_1        #store to i
02: aload_0         #load a to stack
03: arraylength     #get array length
04: istore_2        #store to j
05: iload_1         #load i to stack
06: iload_2         #load j to stack
07: if_icmpge 22    #jump to end if i >= j
10: iload_1         #load i to stack
11: aload_0         #load a to stack
12: iload_1         #load i to stack
13: aaload          #load a[i]
14: invokevirtual length #call length()
17: iadd            #add length and i
18: istore_1        #store to i
19: goto 05
22: return
length():
00: aload_0         #load this to stack
01: getfield count #load count field
04: ireturn         #return it
```

```
class Loop {
 void main(String[] a){
  int i = 0;
  int j = a.length;
  while (i < j) {
   i += a[i].length();}}}
```

1. Adds up lengths.
2. May not terminate.
3. May throw `NullPointerExc`

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0        #load 0 to stack
01: istore_1        #store to i
02: aload_0         #load a to stack
03: arraylength     #get array length
04: istore_2        #store to j
05: iload_1         #load i to stack
06: iload_2         #load j to stack
07: if_icmpge 22    #jump to end if i >= j
10: iload_1         #load i to stack
11: aload_0         #load a to stack
12: iload_1         #load i to stack
13: aaload          #load a[i]
14: invokevirtual length #call length()
17: iadd            #add length and i
18: istore_1        #store to i
19: goto 05
22: return
length():
00: aload_0         #load this to stack
01: getfield count #load count field
04: ireturn         #return it
```

$00 \mid \mathtt{a}:a_1 \mid \varepsilon$
$a_1{:}\mathtt{String}[\,]\ i_1 \qquad i_1{:}[\geq 0]$

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0        #load 0 to stack
01: istore_1        #store to i
02: aload_0         #load a to stack
03: arraylength     #get array length
04: istore_2        #store to j
05: iload_1         #load i to stack
06: iload_2         #load j to stack
07: if_icmpge 22    #jump to end if i >= j
10: iload_1         #load i to stack
11: aload_0         #load a to stack
12: iload_1         #load i to stack
13: aaload          #load a[i]
14: invokevirtual length #call length()
17: iadd            #add length and i
18: istore_1        #store to i
19: goto 05
22: return
length():
00: aload_0         #load this to stack
01: getfield count #load count field
04: ireturn         #return it
```

00 | a:a_1 | ε
a_1:String[] i_1 i_1:[$\geq$0]

stack frame:

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0        #load 0 to stack
01: istore_1        #store to i
02: aload_0         #load a to stack
03: arraylength     #get array length
04: istore_2        #store to j
05: iload_1         #load i to stack
06: iload_2         #load j to stack
07: if_icmpge 22    #jump to end if i >= j
10: iload_1         #load i to stack
11: aload_0         #load a to stack
12: iload_1         #load i to stack
13: aaload          #load a[i]
14: invokevirtual length #call length()
17: iadd            #add length and i
18: istore_1        #store to i
19: goto 05
22: return
length():
00: aload_0         #load this to stack
01: getfield count #load count field
04: ireturn         #return it
```

00 | a:a_1 | ε
a_1:String[] i_1 i_1:[$\geq$0]

stack frame:

- Next program instruction

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0        #load 0 to stack
01: istore_1        #store to i
02: aload_0         #load a to stack
03: arraylength     #get array length
04: istore_2        #store to j
05: iload_1         #load i to stack
06: iload_2         #load j to stack
07: if_icmpge 22    #jump to end if i >= j
10: iload_1         #load i to stack
11: aload_0         #load a to stack
12: iload_1         #load i to stack
13: aaload          #load a[i]
14: invokevirtual length #call length()
17: iadd            #add length and i
18: istore_1        #store to i
19: goto 05
22: return
length():
00: aload_0         #load this to stack
01: getfield count #load count field
04: ireturn         #return it
```

00 | a : a_1 | ε
a_1 : String[] i_1 i_1 : [$\geq$0]

stack frame:

- Next program instruction
- Local variables
- Operand stack

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0        #load 0 to stack
01: istore_1        #store to i
02: aload_0         #load a to stack
03: arraylength     #get array length
04: istore_2        #store to j
05: iload_1         #load i to stack
06: iload_2         #load j to stack
07: if_icmpge 22    #jump to end if i >= j
10: iload_1         #load i to stack
11: aload_0         #load a to stack
12: iload_1         #load i to stack
13: aaload          #load a[i]
14: invokevirtual length #call length()
17: iadd            #add length and i
18: istore_1        #store to i
19: goto 05
22: return
length():
00: aload_0         #load this to stack
01: getfield count #load count field
04: ireturn         #return it
```

$00 \mid \mathtt{a}:a_1 \mid \varepsilon$

$a_1:\mathtt{String}[\,]\ i_1 \qquad i_1:[\geq 0]$

stack frame:

- Next program instruction
- Local variables
- Operand stack

heap information:

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0        #load 0 to stack
01: istore_1        #store to i
02: aload_0         #load a to stack
03: arraylength     #get array length
04: istore_2        #store to j
05: iload_1         #load i to stack
06: iload_2         #load j to stack
07: if_icmpge 22    #jump to end if i >= j
10: iload_1         #load i to stack
11: aload_0         #load a to stack
12: iload_1         #load i to stack
13: aaload          #load a[i]
14: invokevirtual length #call length()
17: iadd            #add length and i
18: istore_1        #store to i
19: goto 05
22: return
length():
00: aload_0         #load this to stack
01: getfield count #load count field
04: ireturn         #return it
```

00 | a : a_1 | ε

a_1 : String[ ] i_1 $\quad$ i_1 : [$\geq$0]

stack frame:

- Next program instruction
- Local variables
- Operand stack

heap information:

- at a_1 is String array content unknown, length is i_1

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0        #load 0 to stack
01: istore_1        #store to i
02: aload_0         #load a to stack
03: arraylength     #get array length
04: istore_2        #store to j
05: iload_1         #load i to stack
06: iload_2         #load j to stack
07: if_icmpge 22    #jump to end if i >= j
10: iload_1         #load i to stack
11: aload_0         #load a to stack
12: iload_1         #load i to stack
13: aaload          #load a[i]
14: invokevirtual length #call length()
17: iadd            #add length and i
18: istore_1        #store to i
19: goto 05
22: return
length():
00: aload_0         #load this to stack
01: getfield count #load count field
04: ireturn         #return it
```

00 | a:a_1 | ε
a_1:String[] i_1 i_1:[$\geq$0]

stack frame:

- Next program instruction
- Local variables
- Operand stack

heap information:

- at a_1 is String array content unknown, length is i_1
- at i_1 is a non-negative integer

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0        #load 0 to stack
01: istore_1        #store to i
02: aload_0         #load a to stack
03: arraylength     #get array length
04: istore_2        #store to j
05: iload_1         #load i to stack
06: iload_2         #load j to stack
07: if_icmpge 22    #jump to end if i >= j
10: iload_1         #load i to stack
11: aload_0         #load a to stack
12: iload_1         #load i to stack
13: aaload          #load a[i]
14: invokevirtual length #call length()
17: iadd            #add length and i
18: istore_1        #store to i
19: goto 05
22: return
length():
00: aload_0         #load this to stack
01: getfield count #load count field
04: ireturn         #return it
```

$00 \mid \mathtt{a}:a_1 \mid \varepsilon$
$a_1:\mathtt{String}[\,]\ i_1 \qquad i_1:[\geq 0]$

stack frame:

- Next program instruction
- Local variables
- Operand stack

heap information:

- at a_1 is `String` array content unknown, length is i_1
- at i_1 is a non-negative integer
- Known `String` object: $\mathtt{String}(\mathtt{count}=i_3,\ldots)$

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0        #load 0 to stack
01: istore_1        #store to i
02: aload_0         #load a to stack
03: arraylength     #get array length
04: istore_2        #store to j
05: iload_1         #load i to stack
06: iload_2         #load j to stack
07: if_icmpge 22    #jump to end if i >= j
10: iload_1         #load i to stack
11: aload_0         #load a to stack
12: iload_1         #load i to stack
13: aaload          #load a[i]
14: invokevirtual length #call length()
17: iadd            #add length and i
18: istore_1        #store to i
19: goto 05
22: return
length():
00: aload_0         #load this to stack
01: getfield count #load count field
04: ireturn         #return it
```

$00 \mid \mathtt{a}:a_1 \mid \varepsilon$
$a_1:\mathtt{String}[\,]\ i_1 \qquad i_1:[\geq 0]$

stack frame:

- Next program instruction
- Local variables
- Operand stack

heap information:

- at a_1 is `String` array content unknown, length is i_1
- at i_1 is a non-negative integer
- Known `String` object: $\mathtt{String}(\mathtt{count}=i_3,\ldots)$
- Unknown `String` object: `String(?)`

Abstract JAVA virtual machine states

```
main(String[] a):
00: iconst_0        #load 0 to stack
01: istore_1        #store to i
02: aload_0         #load a to stack
03: arraylength     #get array length
04: istore_2        #store to j
05: iload_1         #load i to stack
06: iload_2         #load j to stack
07: if_icmpge 22    #jump to end if i >= j
10: iload_1         #load i to stack
11: aload_0         #load a to stack
12: iload_1         #load i to stack
13: aaload          #load a[i]
14: invokevirtual length #call length()
17: iadd            #add length and i
18: istore_1        #store to i
19: goto 05
22: return
length():
00: aload_0         #load this to stack
01: getfield count #load count field
04: ireturn         #return it
```

$00 \mid \mathtt{a}:a_1 \mid \varepsilon$
$a_1:\mathtt{String}[\,]\ i_1 \qquad i_1:[\geq 0]$

stack frame:

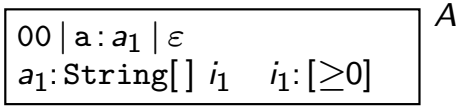
- Next program instruction
- Local variables
- Operand stack

heap information:

- at a_1 is `String` array content unknown, length is i_1
- at i_1 is a non-negative integer
- Known `String` object: $\mathtt{String}(\mathtt{count}=i_3,\ldots)$
- Unknown `String` object: $\mathtt{String}(?)$

Only explicit sharing

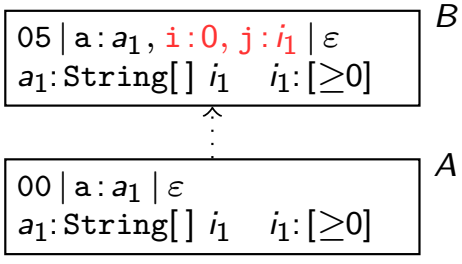
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



State A:

- What can happen when evaluating `main`?
- a_1: Unknown array of `String` objects
- i_1: a_1's unknown length

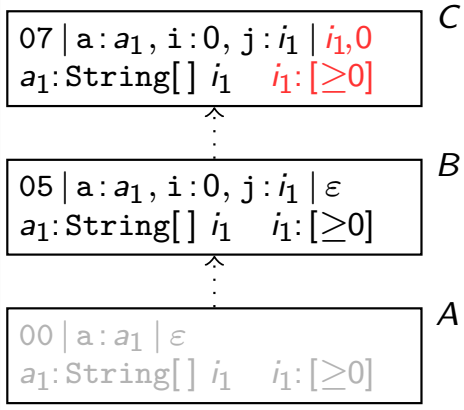
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



State B:

- Stored 0 to i
- Stored arraylength i_1 to j
- *Evaluations* from A to B

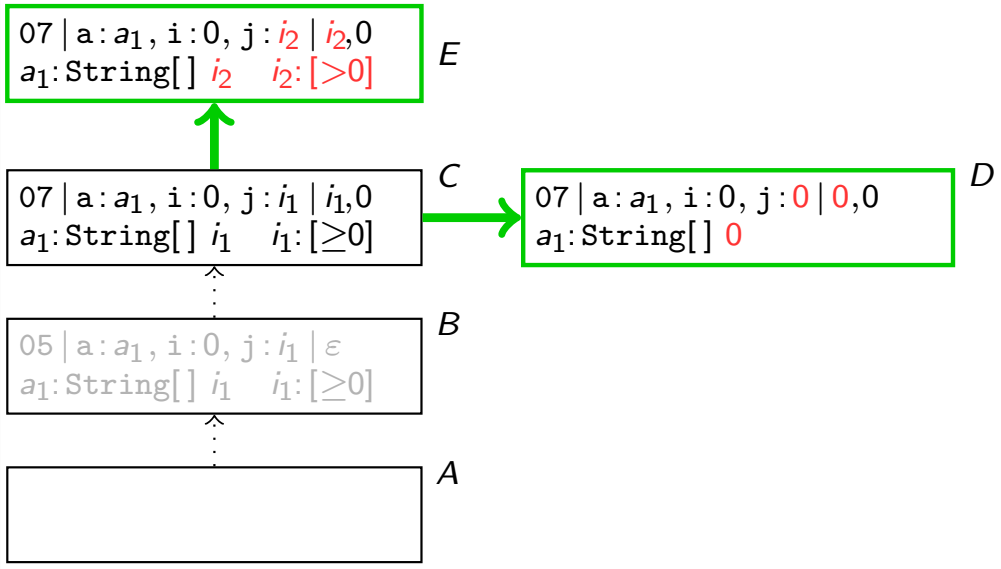
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



State C:

- Load `i` (0) and `j` (i_1) to operand stack
- `if_icmpge` cannot be evaluated

```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



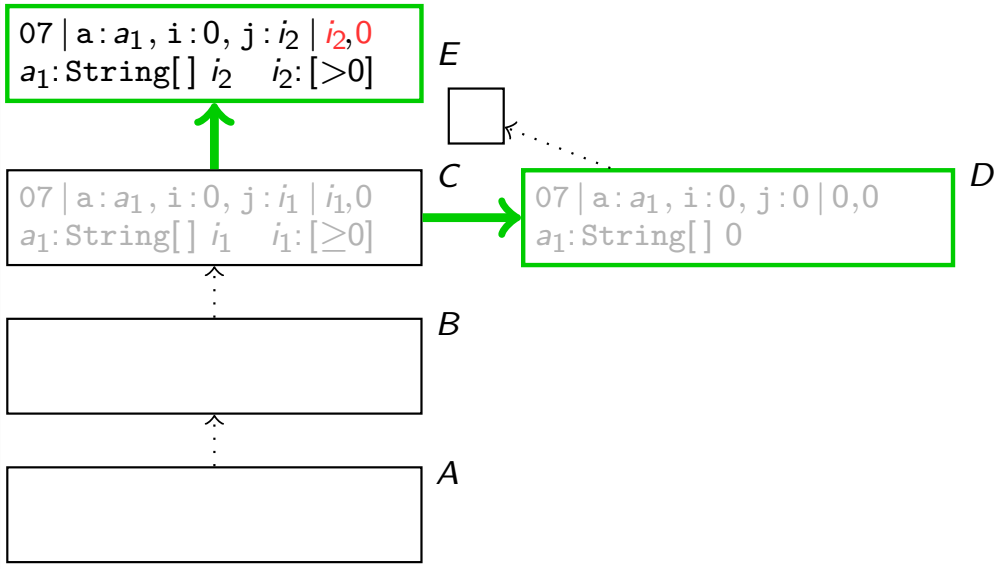
State C, D, E:

- Load `i` (0) and `j` (i_1) to operand stack
- `if_icmpge` cannot be evaluated yet:

$\Rightarrow$ *Refine* information:

- In D, consider case $i_1 = 0$
- In E, consider case $i_1 > 0$

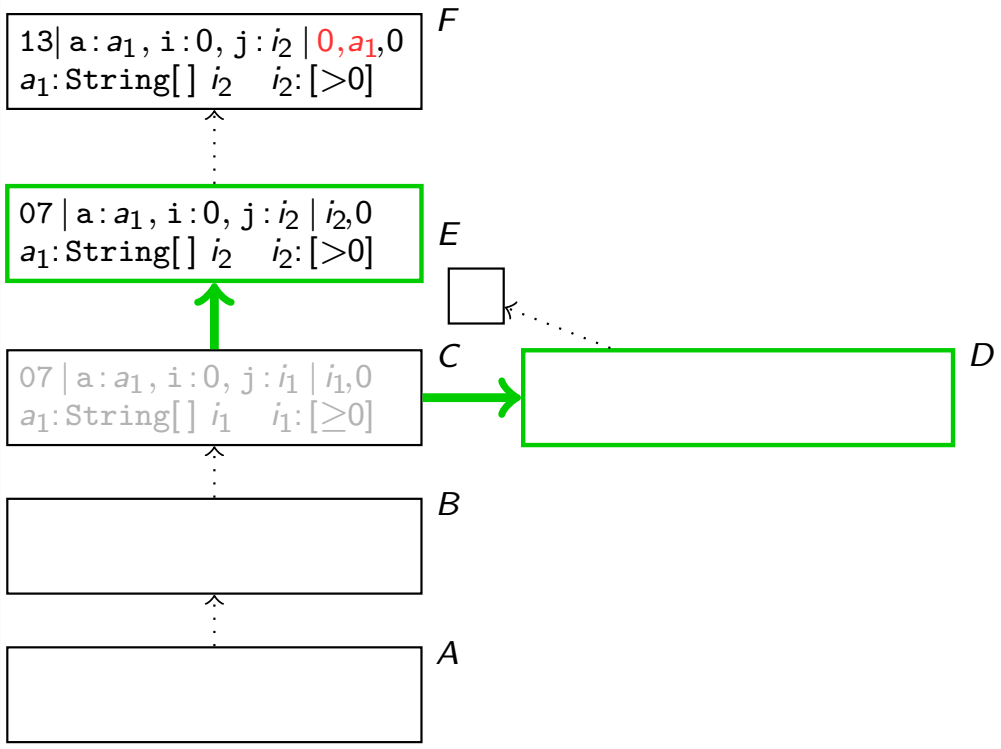
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



States D, E, F:

- D jumps to end of method

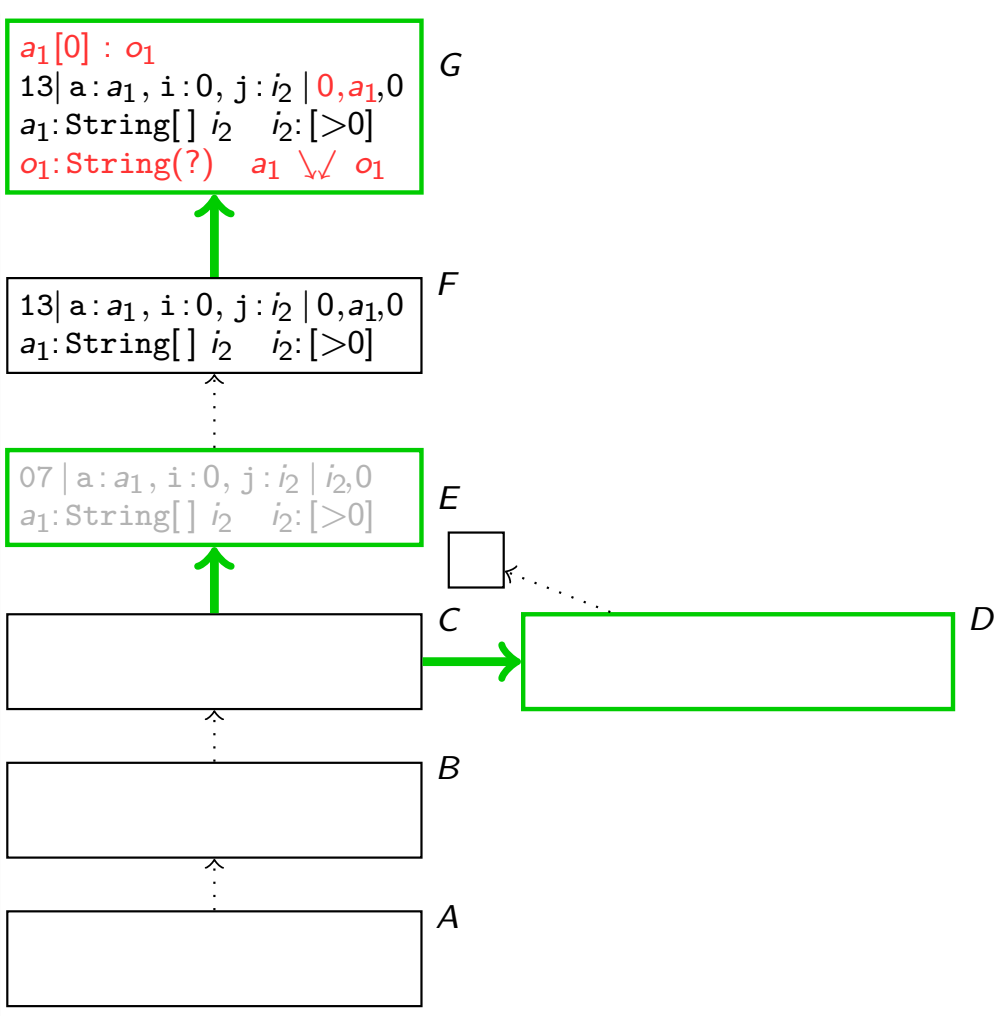
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



States D, E, F:

- D jumps to end of method
- E evaluates to F
- Loads array a (a_1) and index i (0) to stack
- Cannot evaluate aaload: $a_1[0]$ not known

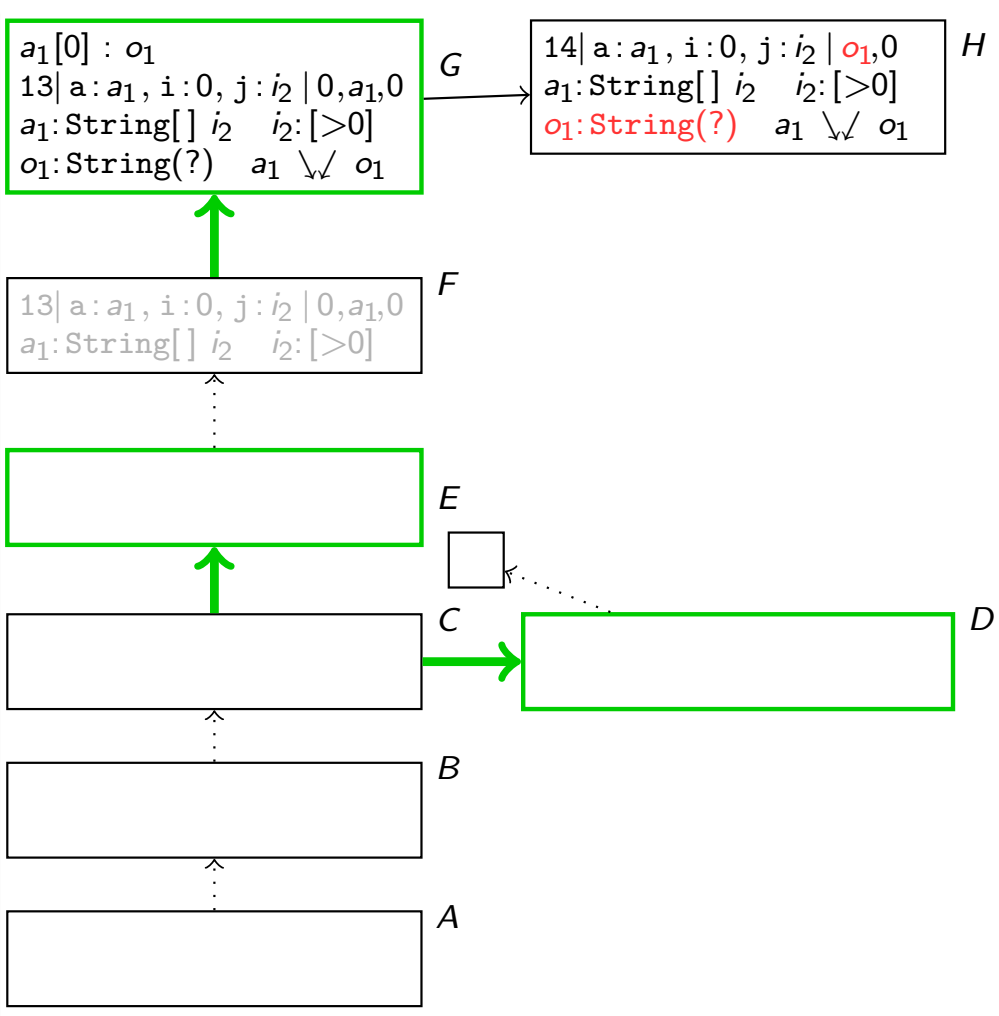
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



State G:

- *Refinement* of F
- o_1 created: `null` or unknown `String`
- $a_1 \curlyvee o_1$: They *share*
- o_1 is value at $a_1[i_2]$

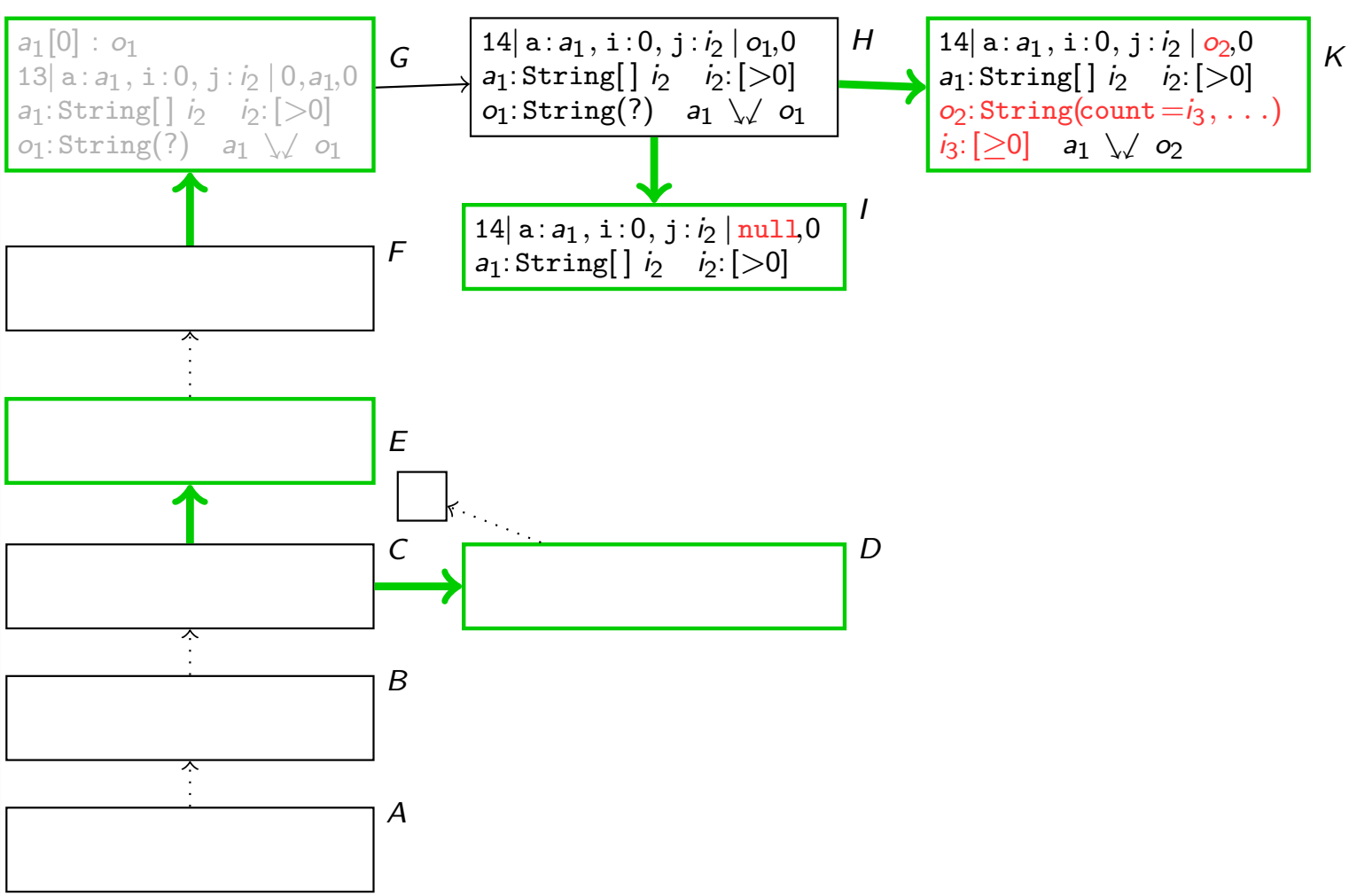
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



States H to K:

- H evaluated from G, loaded o_1 to stack
- `invoke` may throw `NullPointerException`

```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



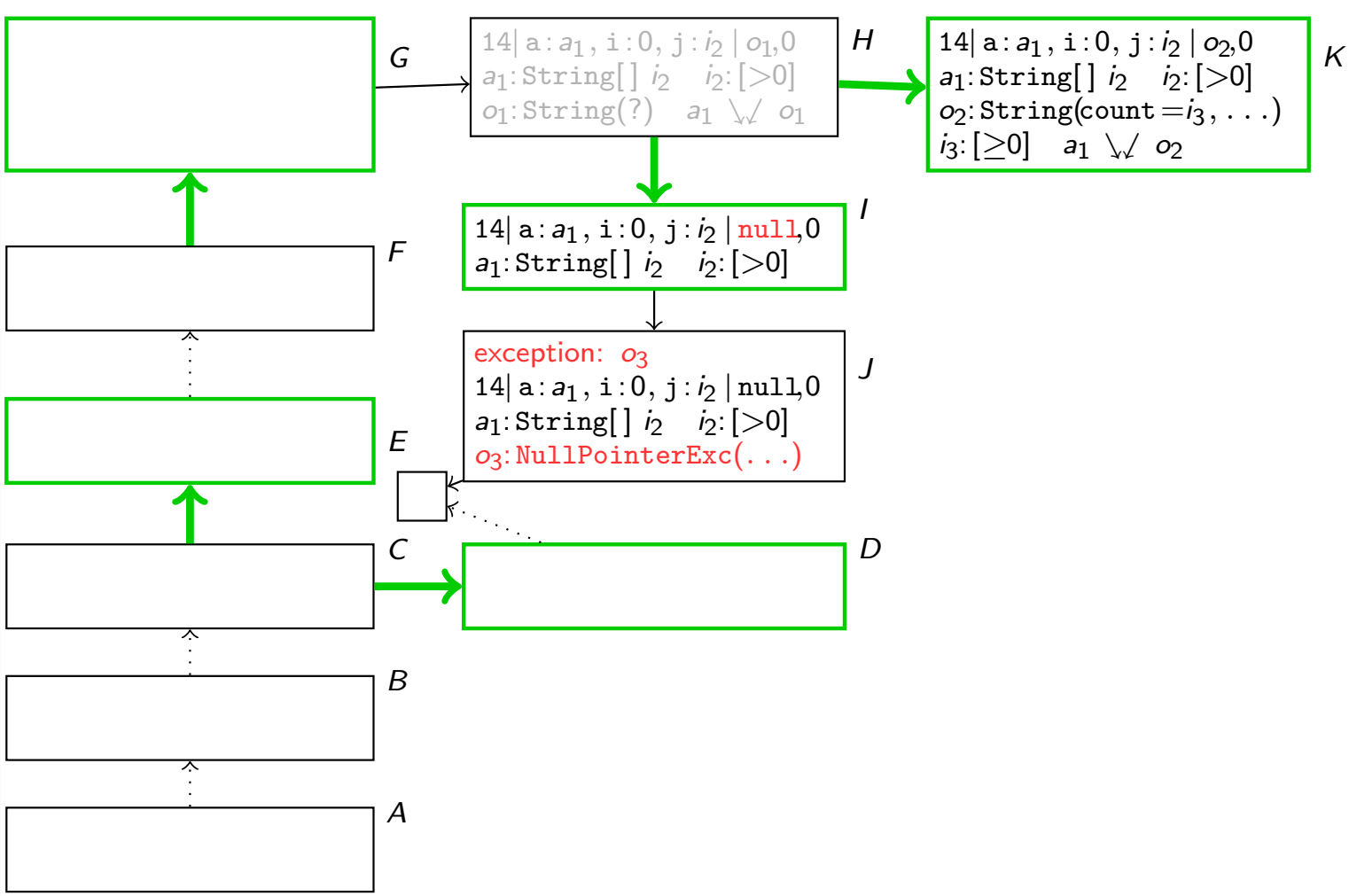
States H to K:

- H evaluated from G, loaded o_1 to stack
- `invoke` may throw `NullPointerException`

⇒ *Refinement*:

- I: Case o_1 is `null`
- K: Case o_1 is some object with fields

```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



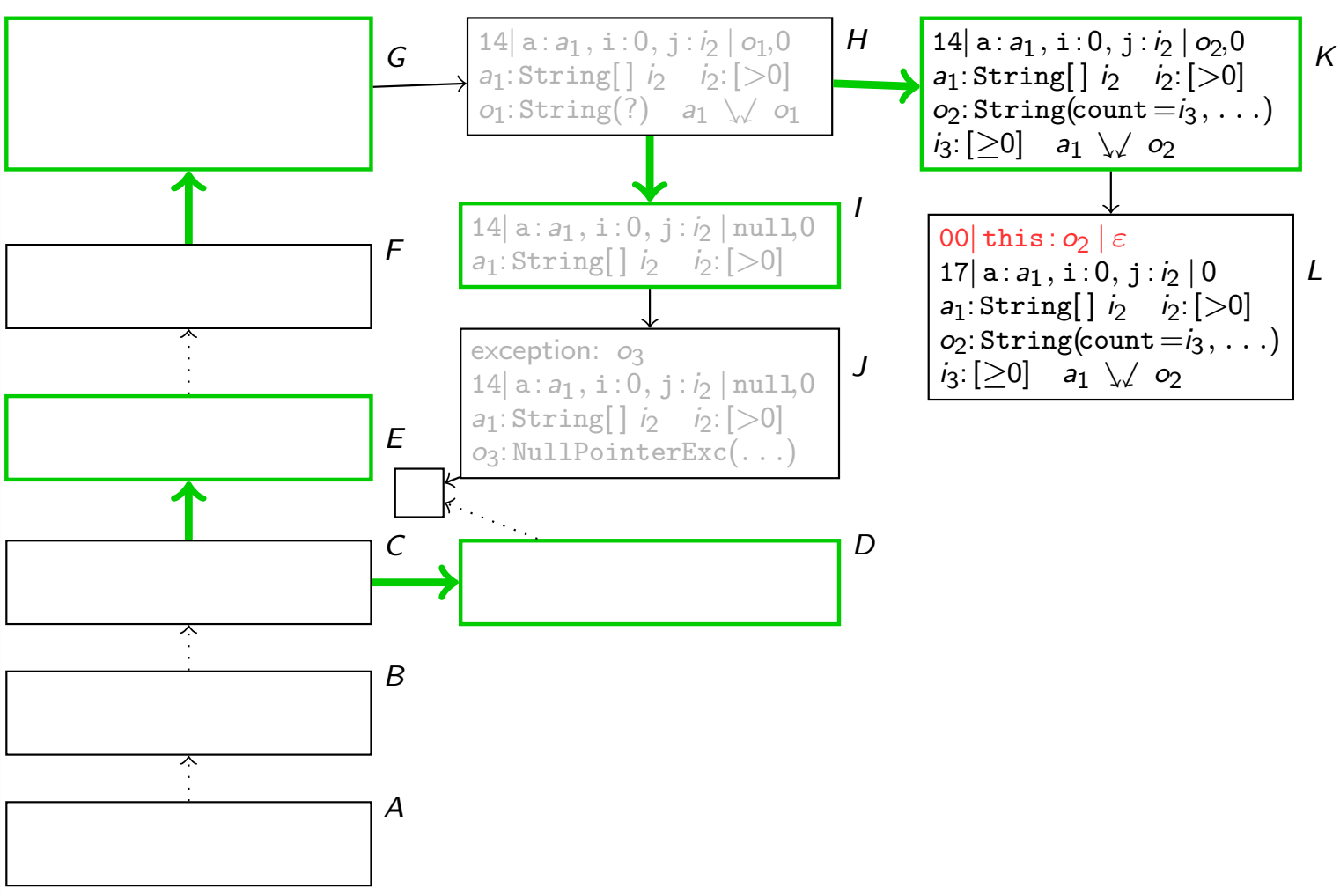
States H to K:

- H evaluated from G, loaded o_1 to stack
- `invoke` may throw `NullPointerException`

$\Rightarrow$ *Refinement*:

- I: Case o_1 is `null` (leads to `NPE`)
- K: Case o_1 is some object with fields

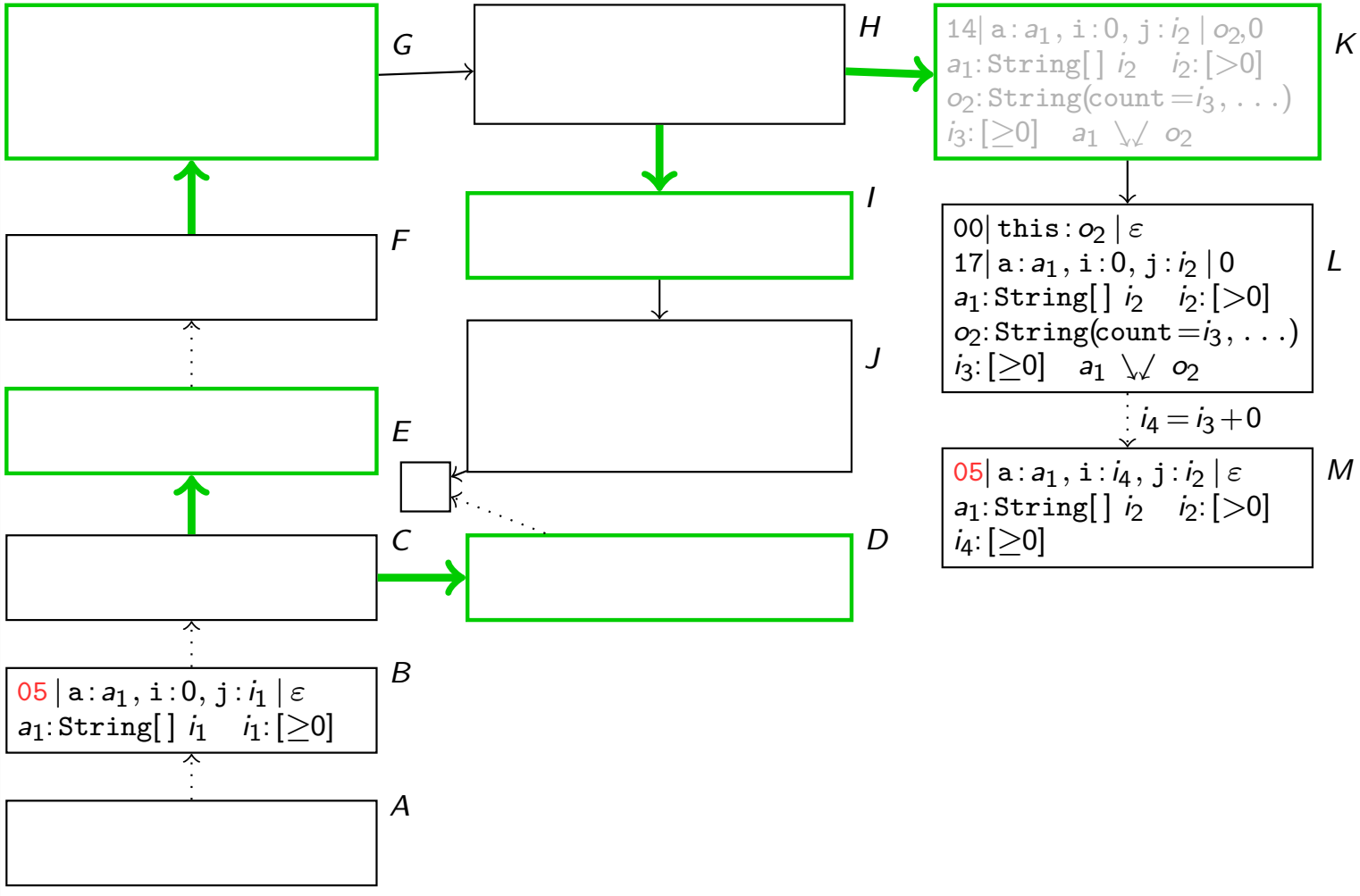
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



State L, M, N:

- Evaluation to L: New stack frame on top

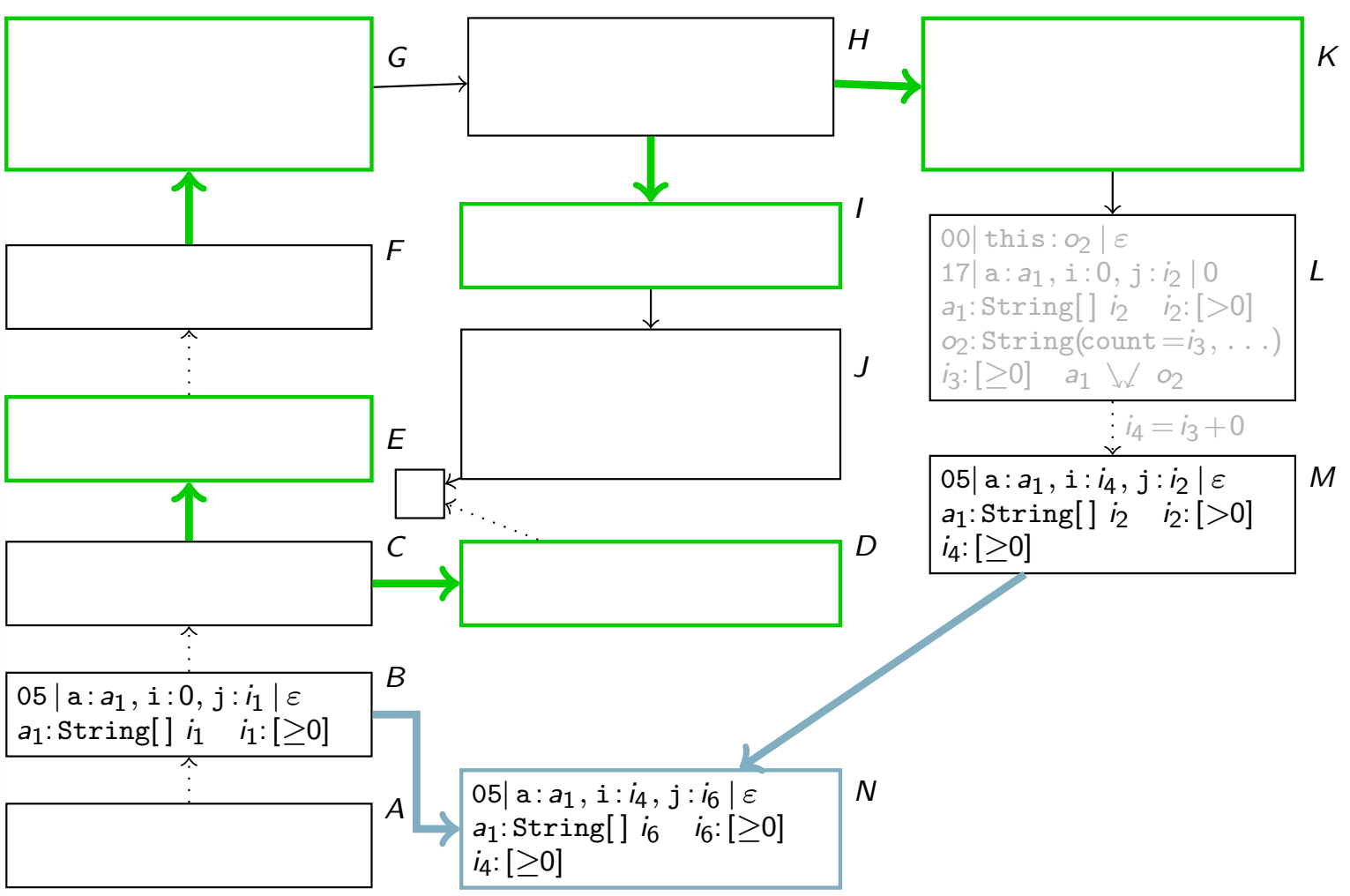
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



State L, M, N:

- Evaluation to L: New stack frame on top
- Evaluation to M: Retrieve length, add to `i`

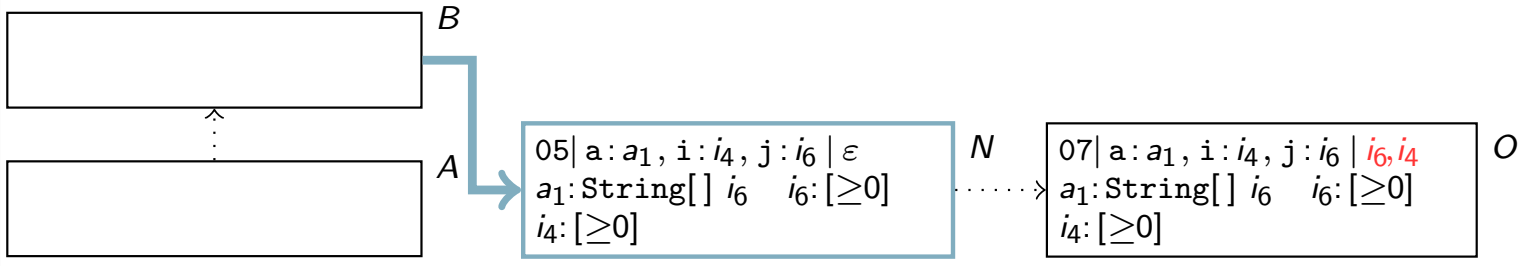
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



State L, M, N:

- Evaluation to L: New stack frame on top
- Evaluation to M: Retrieve length, add to i
- B and M similar: *Merge* states, get N
- N represents both B and M (*instances* of N)

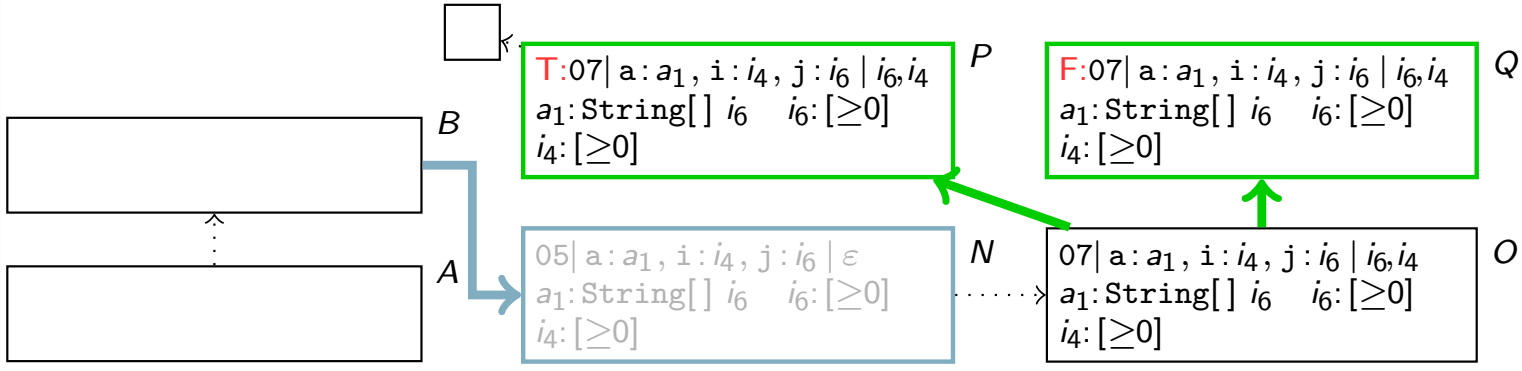
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



States O to S:

- Evaluate to O: Load `i`, `j` to stack

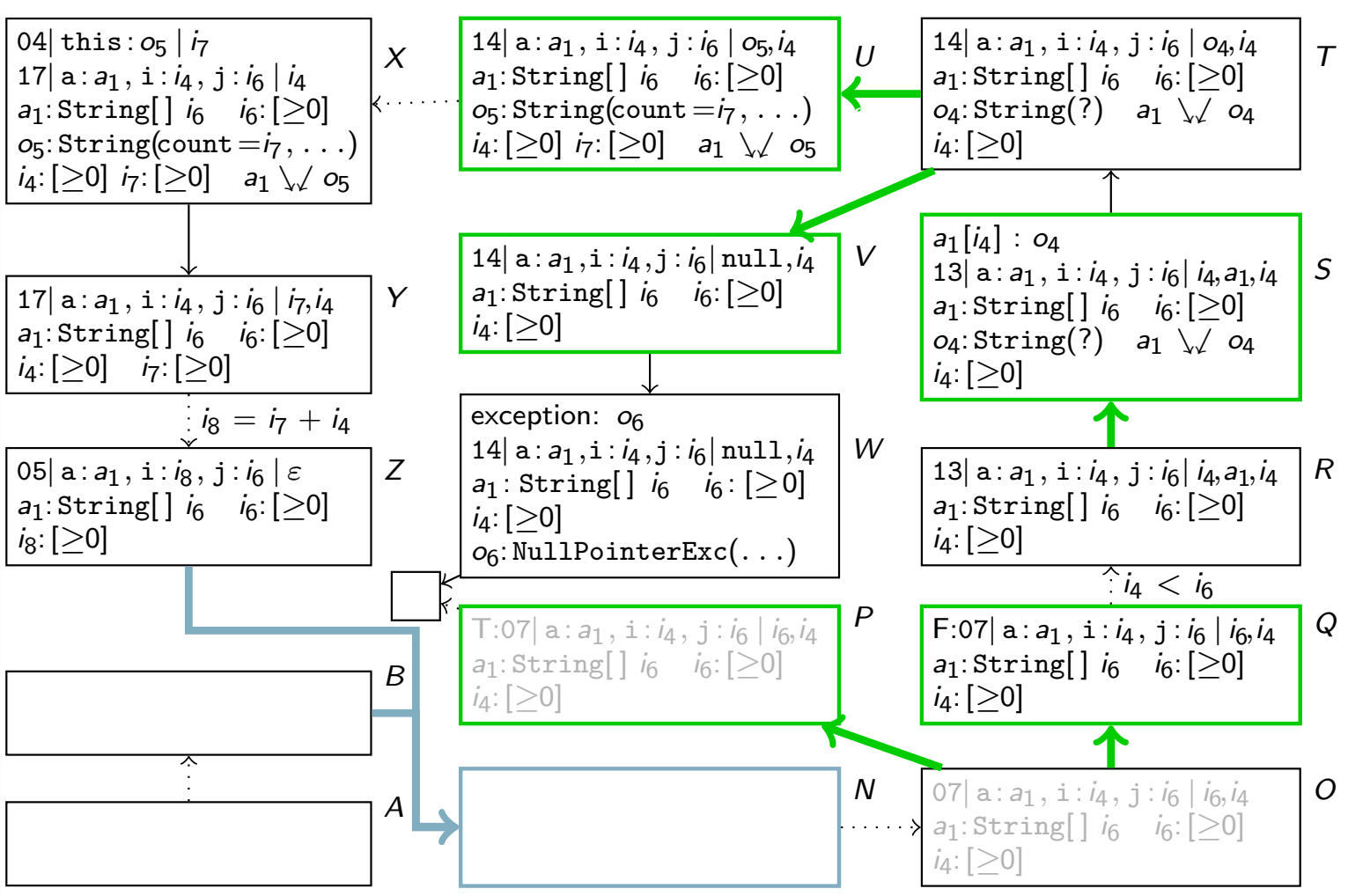
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



States O to S:

- Evaluate to O: Load i, j to stack
- Refine O to P, Q: Decide result of if_icmpge

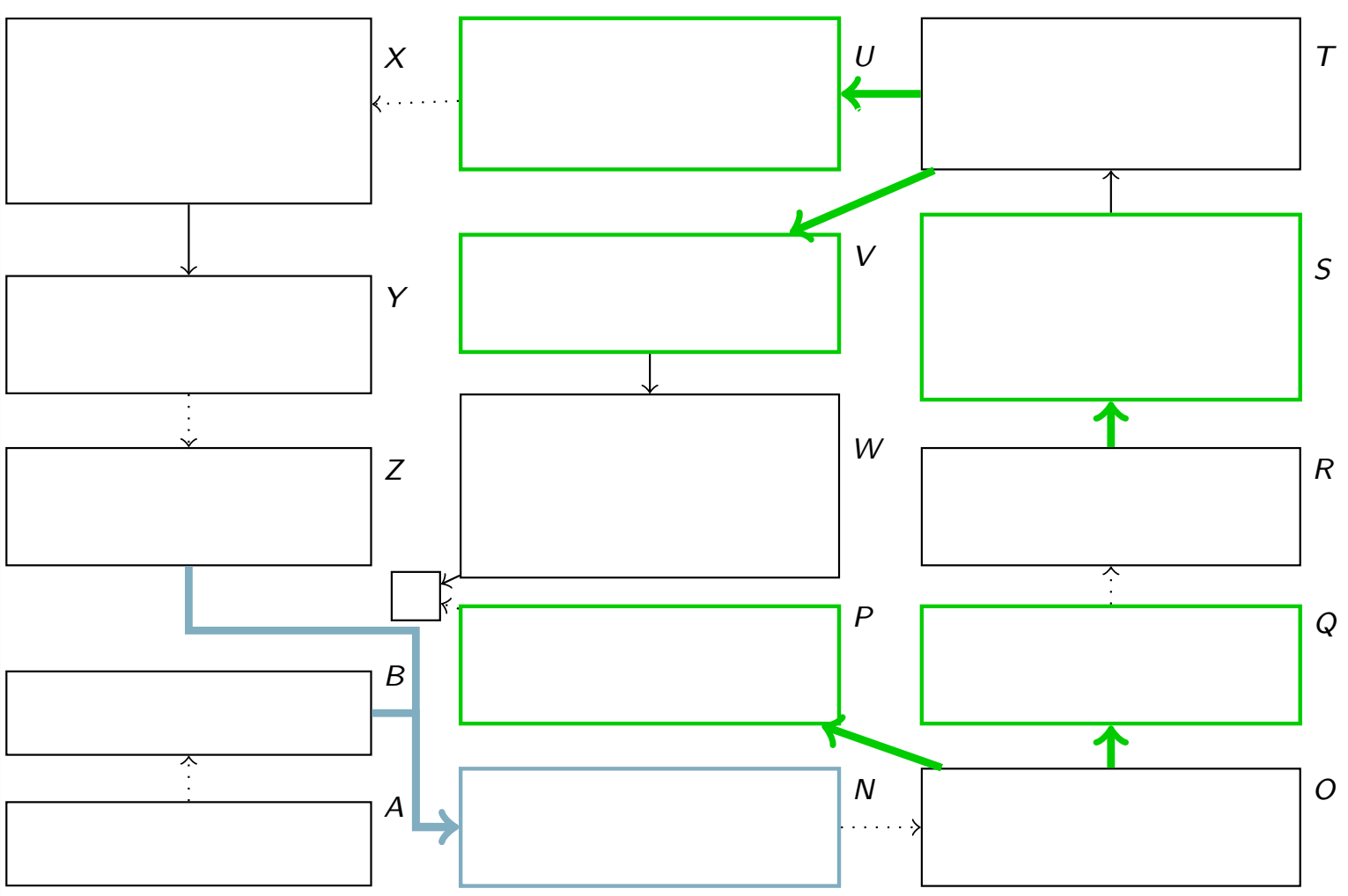
```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



States O to S:

- Evaluate to O: Load `i`, `j` to stack
- Refine O to P, Q: Decide result of `if_icmpge`
- Rest as before

```
 main(String[] a):
00: iconst_0
01: istore_1
02: aload_0
03: arraylength
04: istore_2
05: iload_1
06: iload_2
07: if_icmpge 22
10: iload_1
11: aload_0
12: iload_1
13: aaload
14: invoke length
17: iadd
18: istore_1
19: goto 05
22: return
 length():
00: aload_0
01: getfield count
04: ireturn
```



States O to S:

- Evaluate to O: Load `i`, `j` to stack
- Refine O to P, Q: Decide result of `if_icmpge`
- Rest as before
- All leaves program ends $\Rightarrow$ construction finished

Termination graph construction

1. Choose (abstract) start state

Termination graph construction

1. Choose (abstract) start state
2. Perform
   - symbolic evaluation
   - information refinement
   - instantiation (may need to *merge* states)

   until all leaves are program ends

Termination graph construction

1. Choose (abstract) start state
2. Perform
   - symbolic evaluation
   - information refinement
   - instantiation (may need to *merge* states)

   until all leaves are program ends
3. Recursion and modularity supported (cf. RTA'11)

Witness generation

- Termination graphs are *over*approximations of all runs

Witness generation

- Termination graphs are *over*approximations of all runs
- Let state e occur in graph starting in s
- e really occurring in runs from s?

Witness generation

- Termination graphs are *over*approximations of all runs
- Let state e occur in graph starting in s
- e really occurring in runs from s?

$\Rightarrow$ Generate a witness run

Witness generation

- Termination graphs are *over*approximations of all runs
- Let state e occur in graph starting in s
- e really occurring in runs from s?

$\Rightarrow$ Generate a witness run

Definition

w *witness* for $e \Leftrightarrow$ all runs from w lead to e

Witness generation

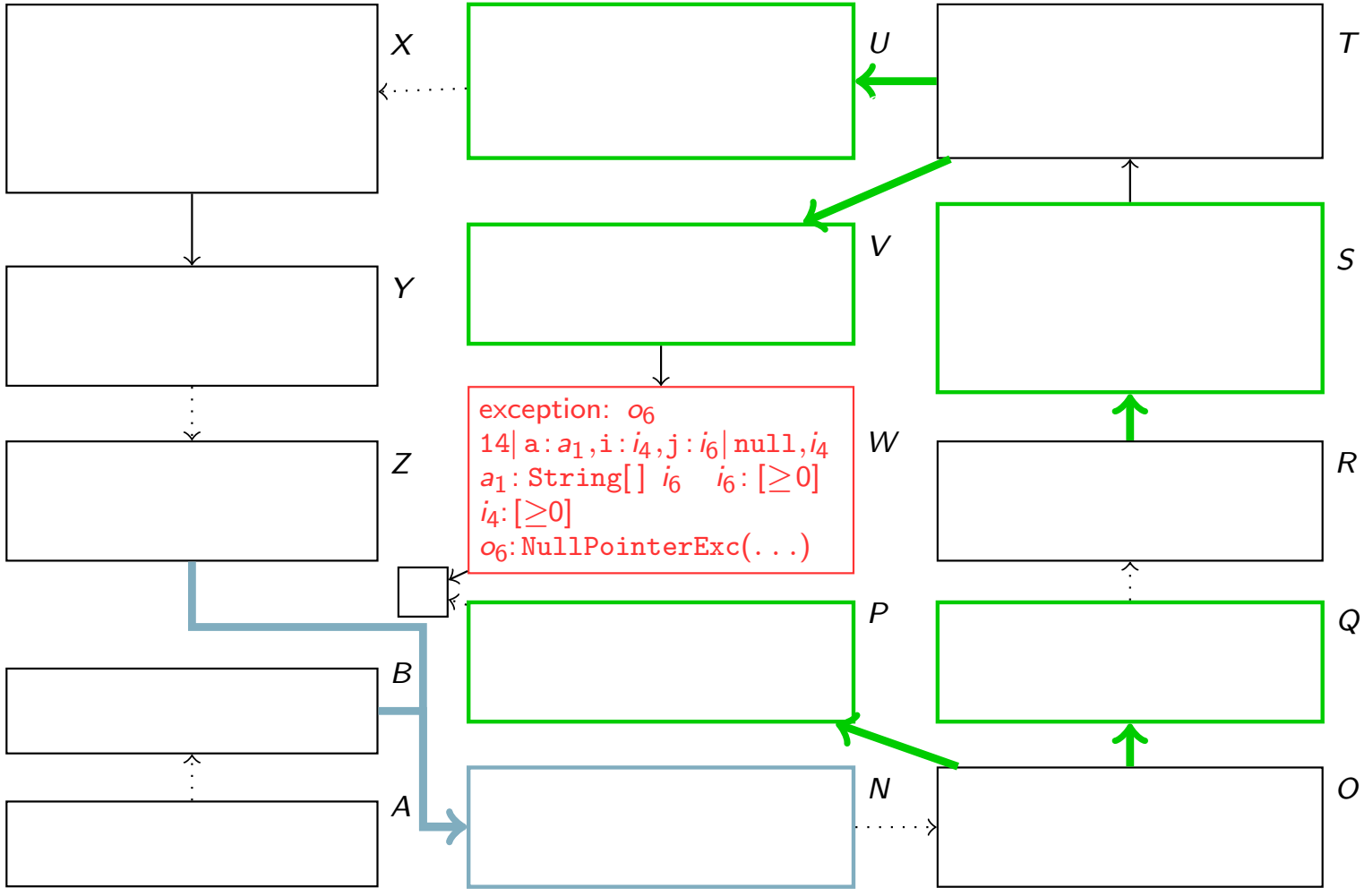
- Termination graphs are *over*approximations of all runs
- Let state e occur in graph starting in s
- e really occurring in runs from s?

$\Rightarrow$ Generate a witness run

Definition

w *witness* for $e \Leftrightarrow$ all runs from w lead to e

Re-use graph: Traverse edges backwards

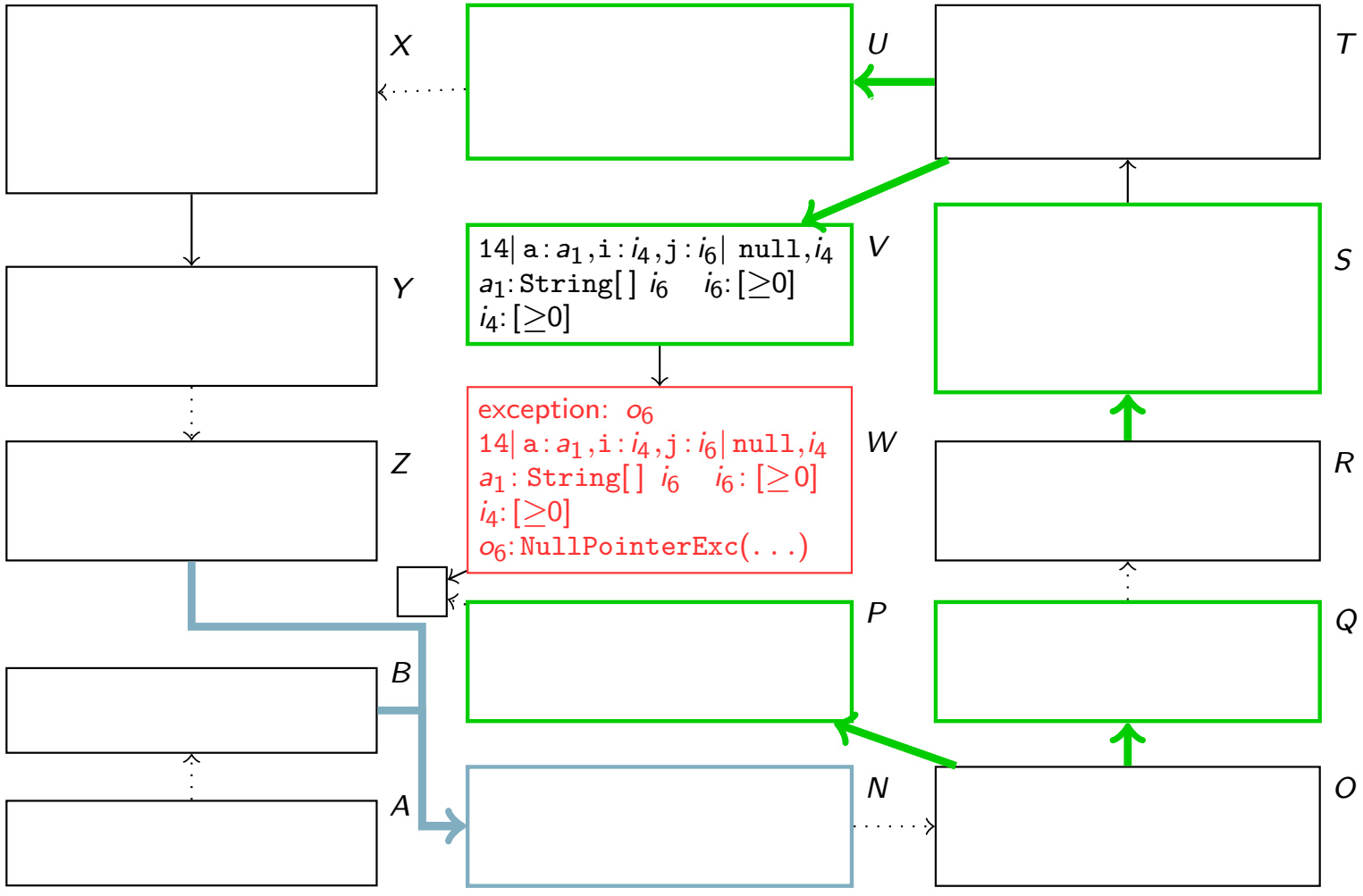


- W: program ends with a `NullPointerException`

$\Rightarrow$ Find witness

Witness for W

exception: o_6
$14|\,\mathtt{a}:a_1,\mathtt{i}:i_4,\mathtt{j}:i_6|\,\mathtt{null},i_4$
a_1 : String[ ] i_6 i_6 : $[\geq 0]$
i_4: $[\geq 0]$
o_6: NullPointerExc(. . .)

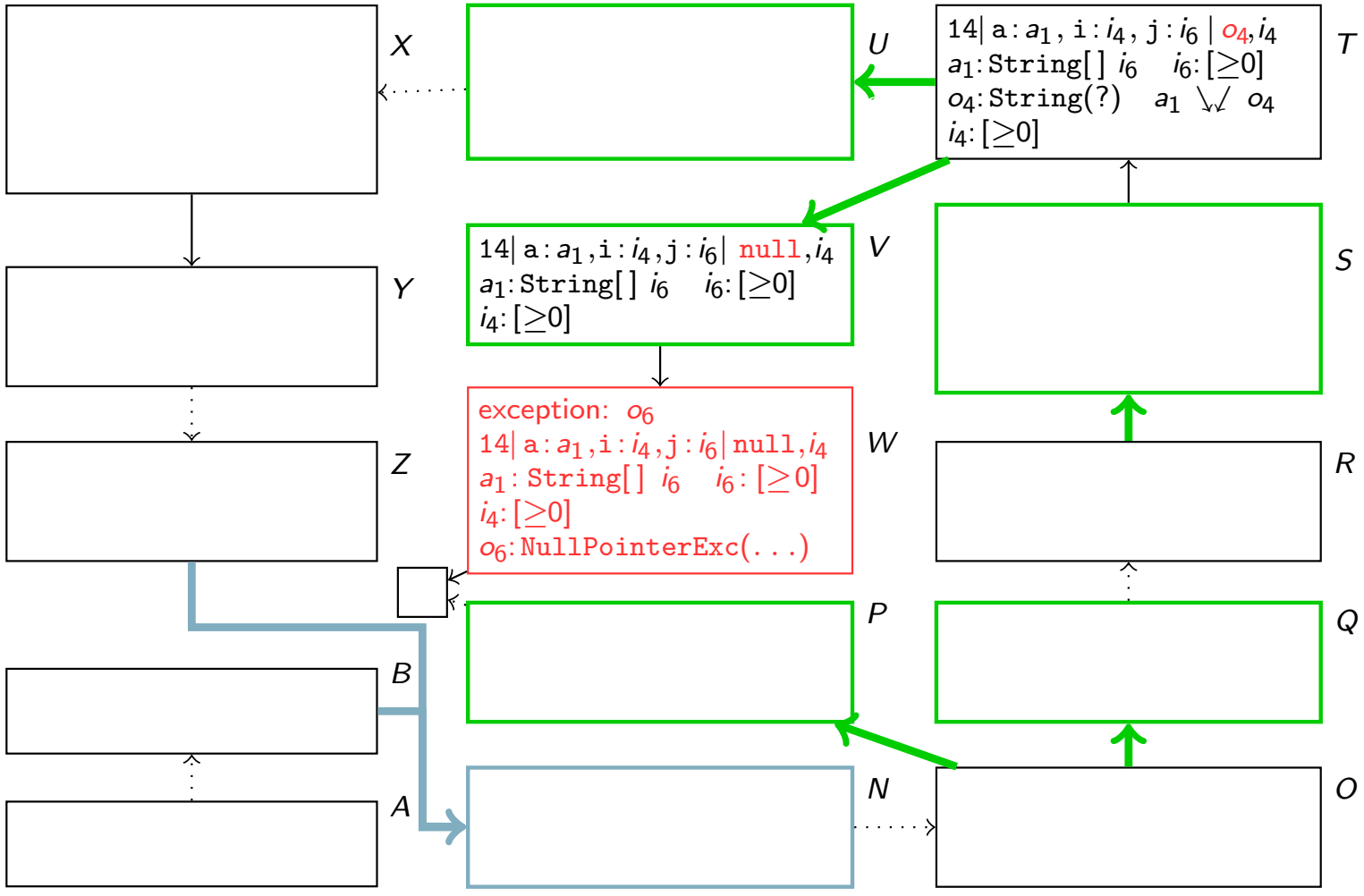


- V evaluated to W

Witness for W

exception: o_6
14| a : a_1 , i : i_4 , j : i_6 | null , i_4
a_1 : String[ ] i_6 i_6 : [≥0]
i_4 : [≥0]
o_6 : NullPointerExc(. . .)

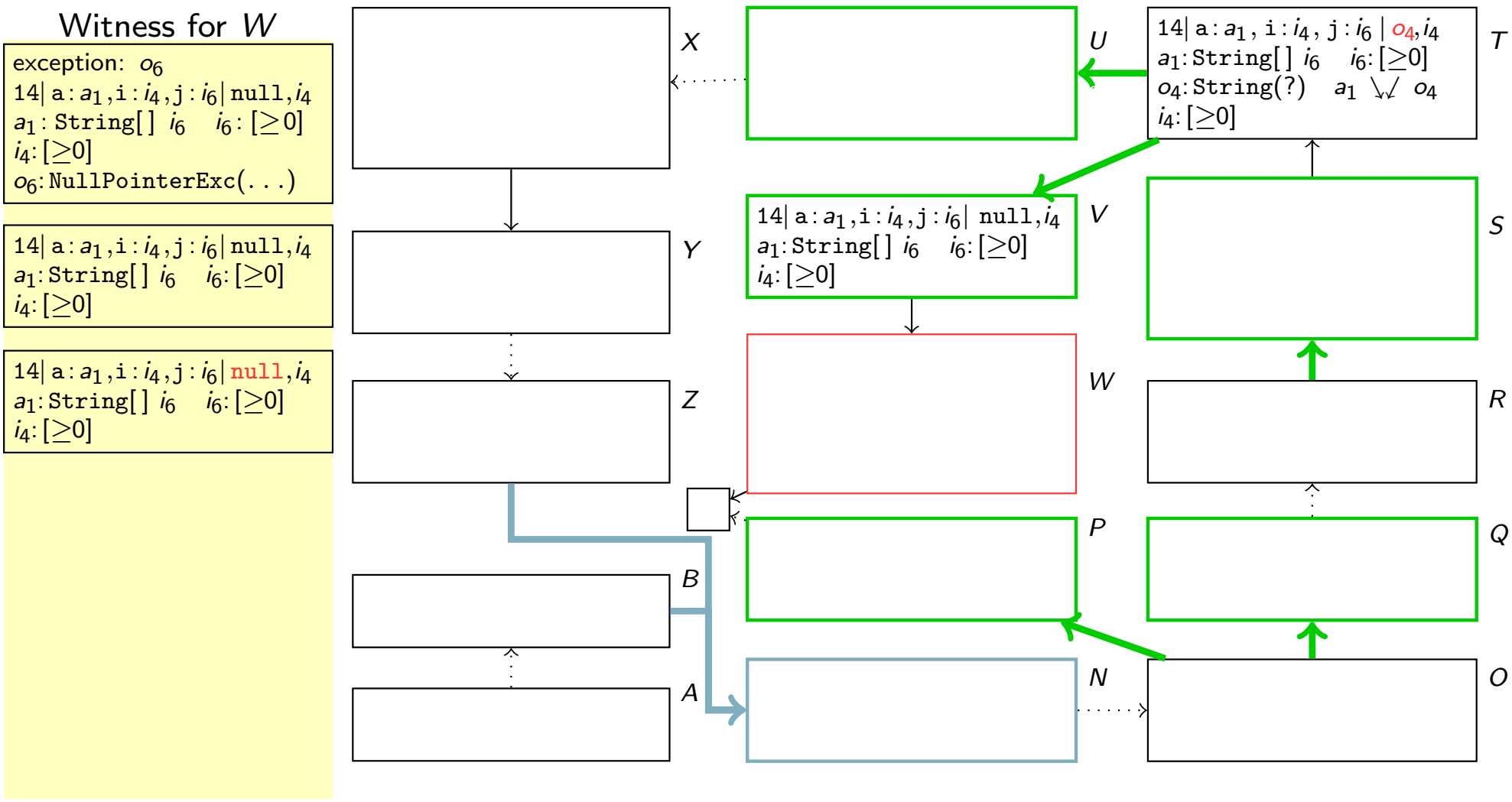
14| a : a_1 , i : i_4 , j : i_6 | null , i_4
a_1 : String[ ] i_6 i_6 : [≥0]
i_4 : [≥0]



- V evaluated to W

$\Rightarrow$ Reverse evaluation step

- T refined to V

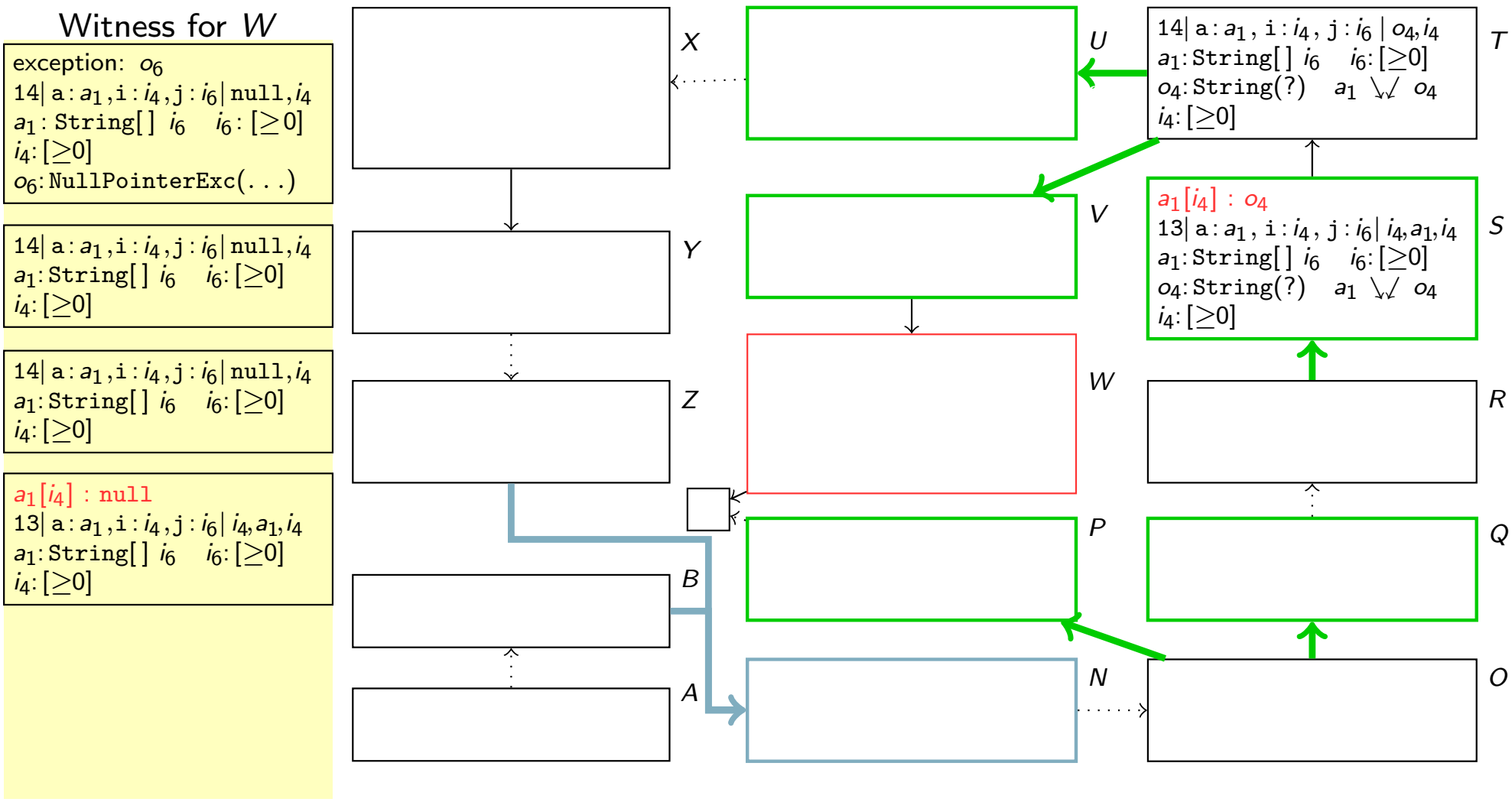


- V evaluated to W

$\Rightarrow$ Reverse evaluation step

- T refined to V

$\Rightarrow$ Re-use current witness



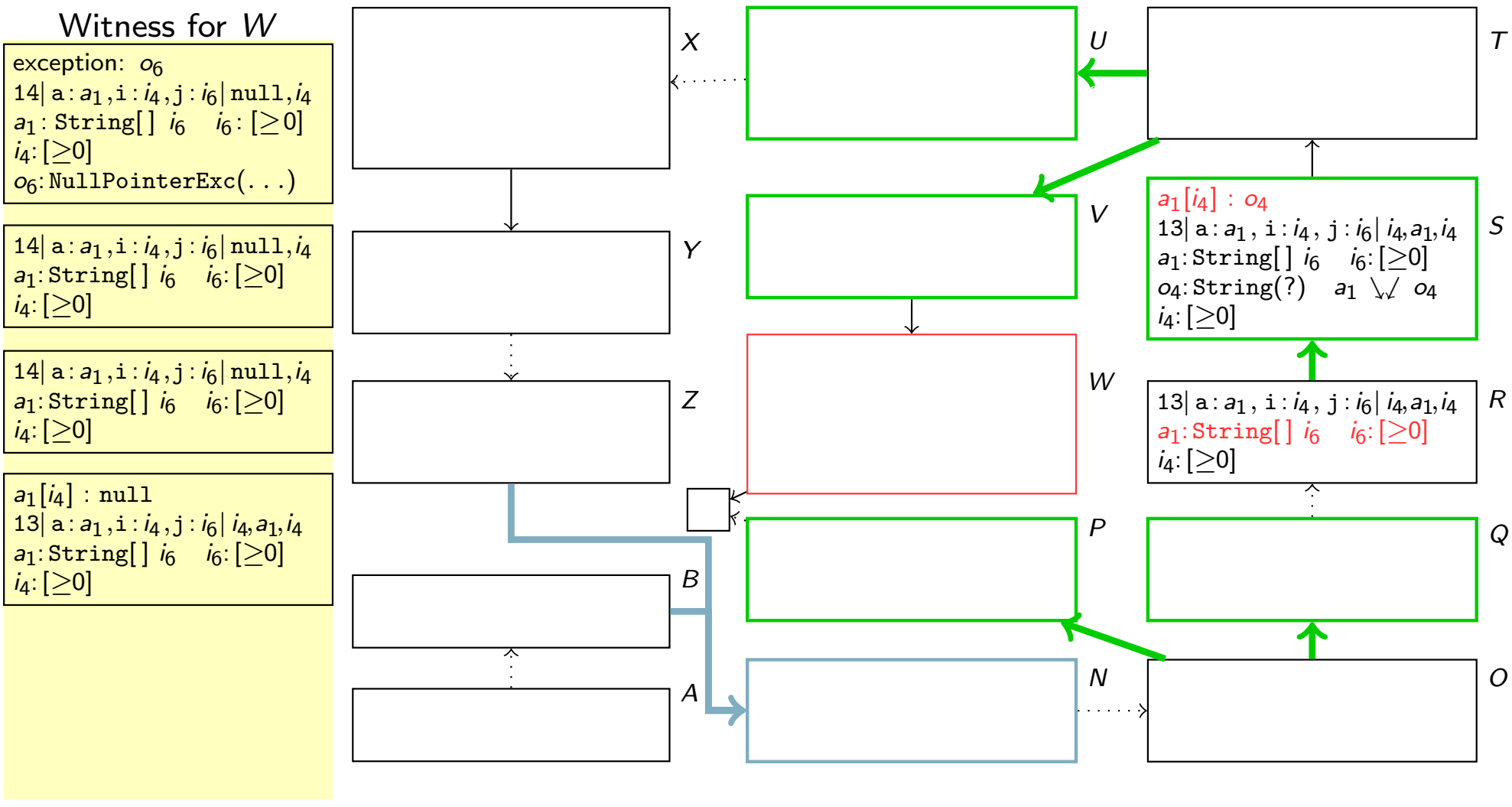
- V evaluated to W

$\Rightarrow$ Reverse evaluation step

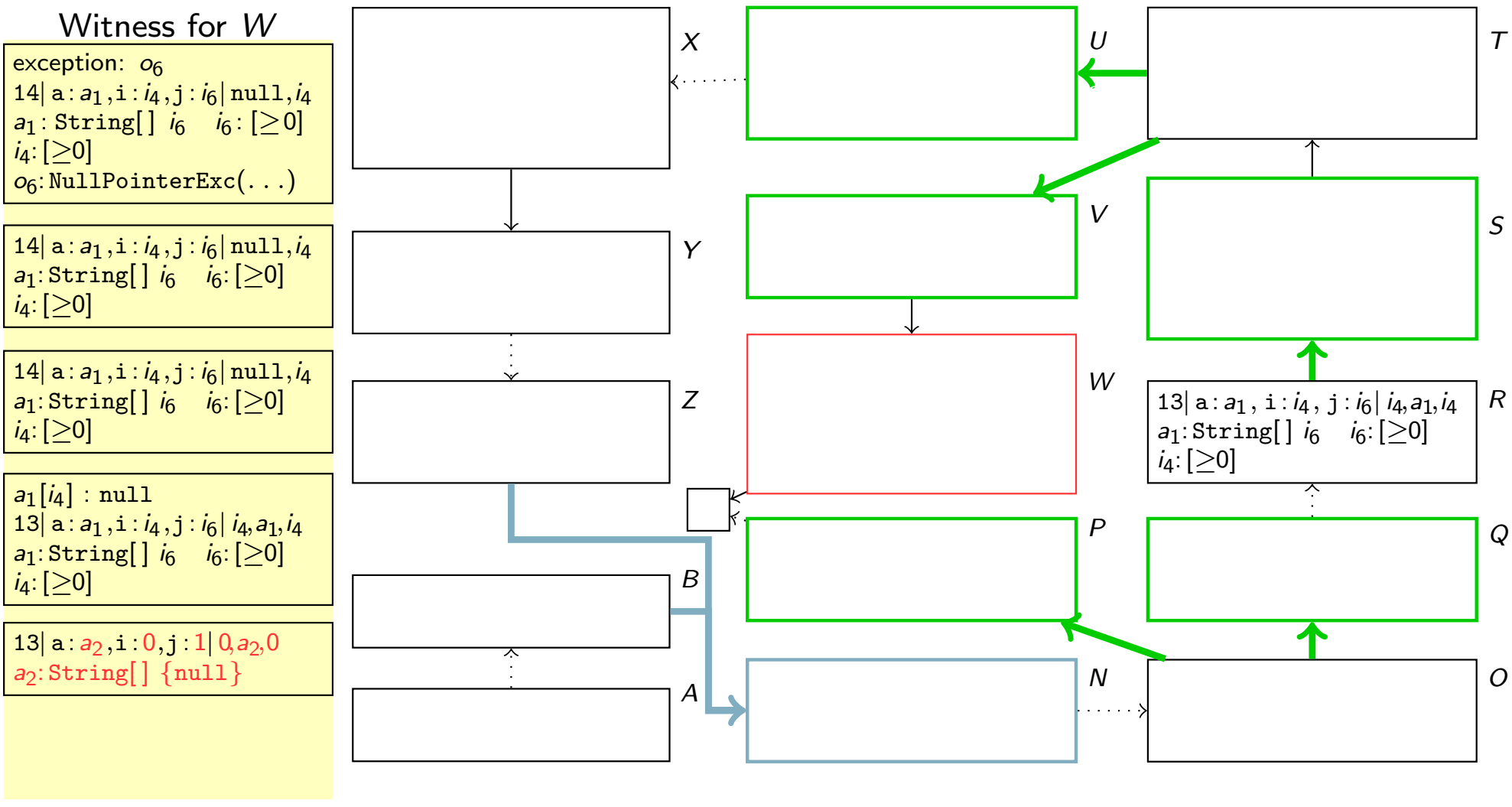
- T refined to V

$\Rightarrow$ Re-use current witness

- S evaluated to T: Reverse evaluation

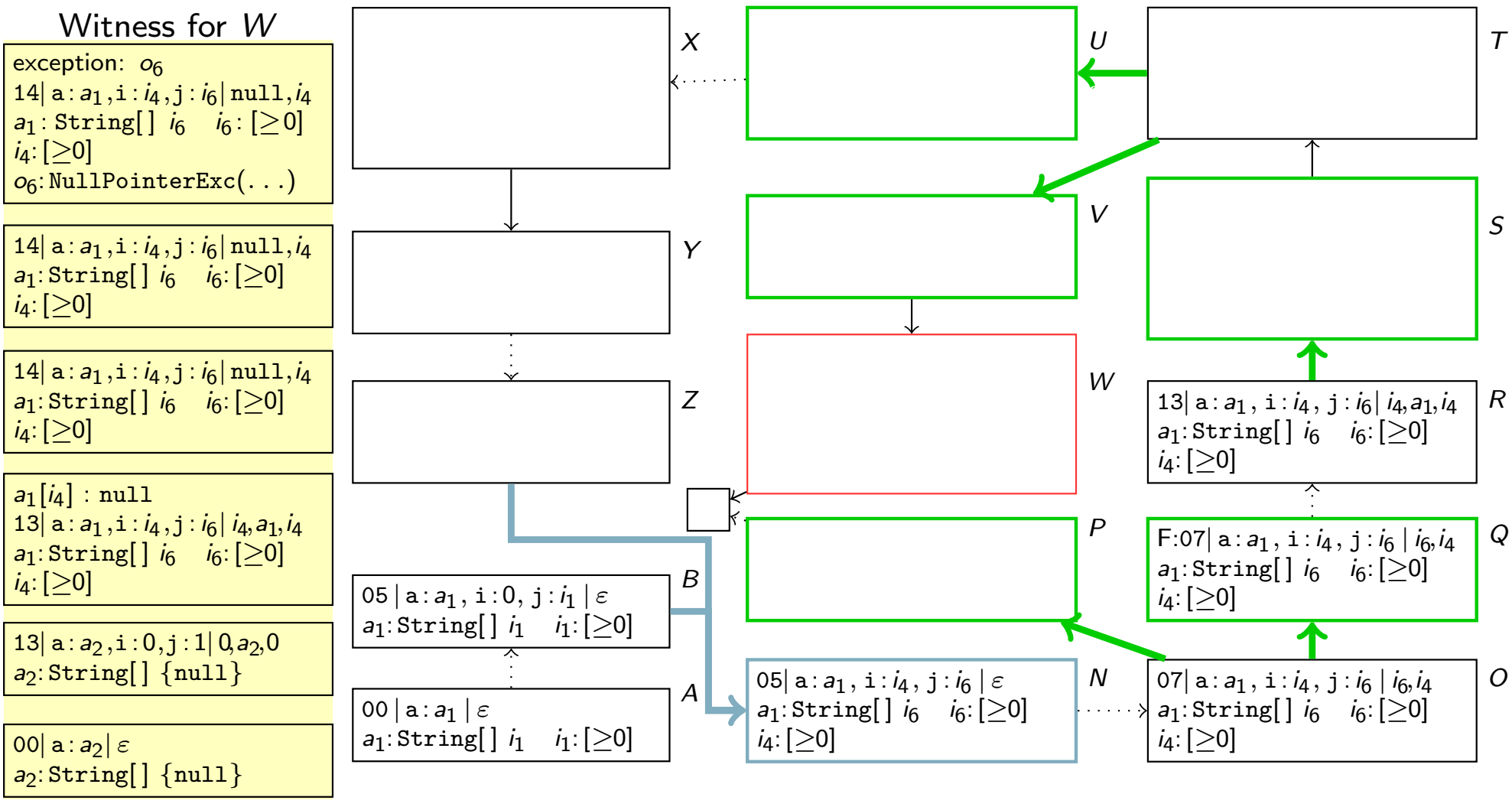


- R refined to S: Re-use current witness



- R refined to S: Re-use current witness - No!
- a_1 too abstract to keep information

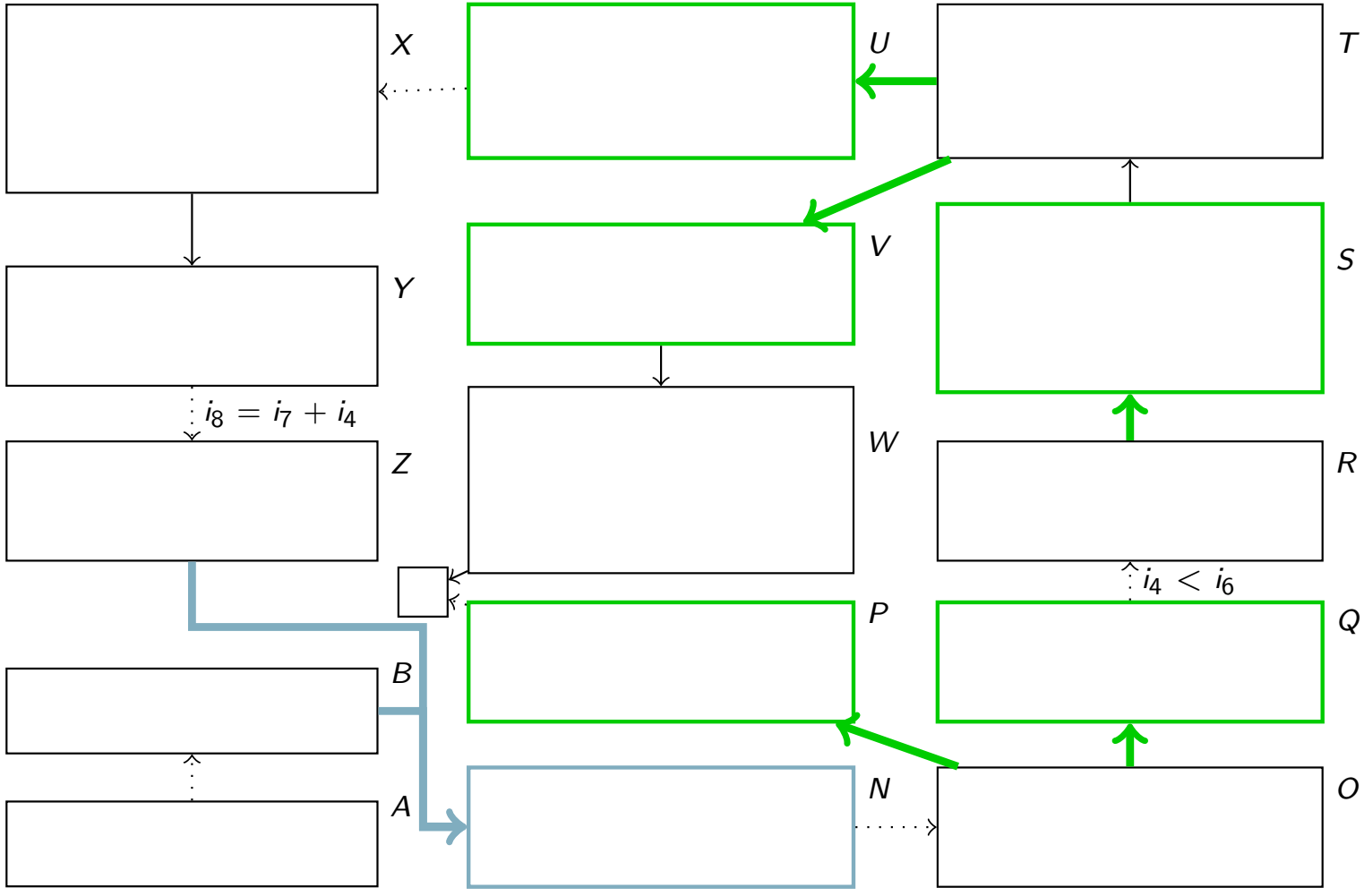
$\Rightarrow$ Instantiate array and index by concrete values



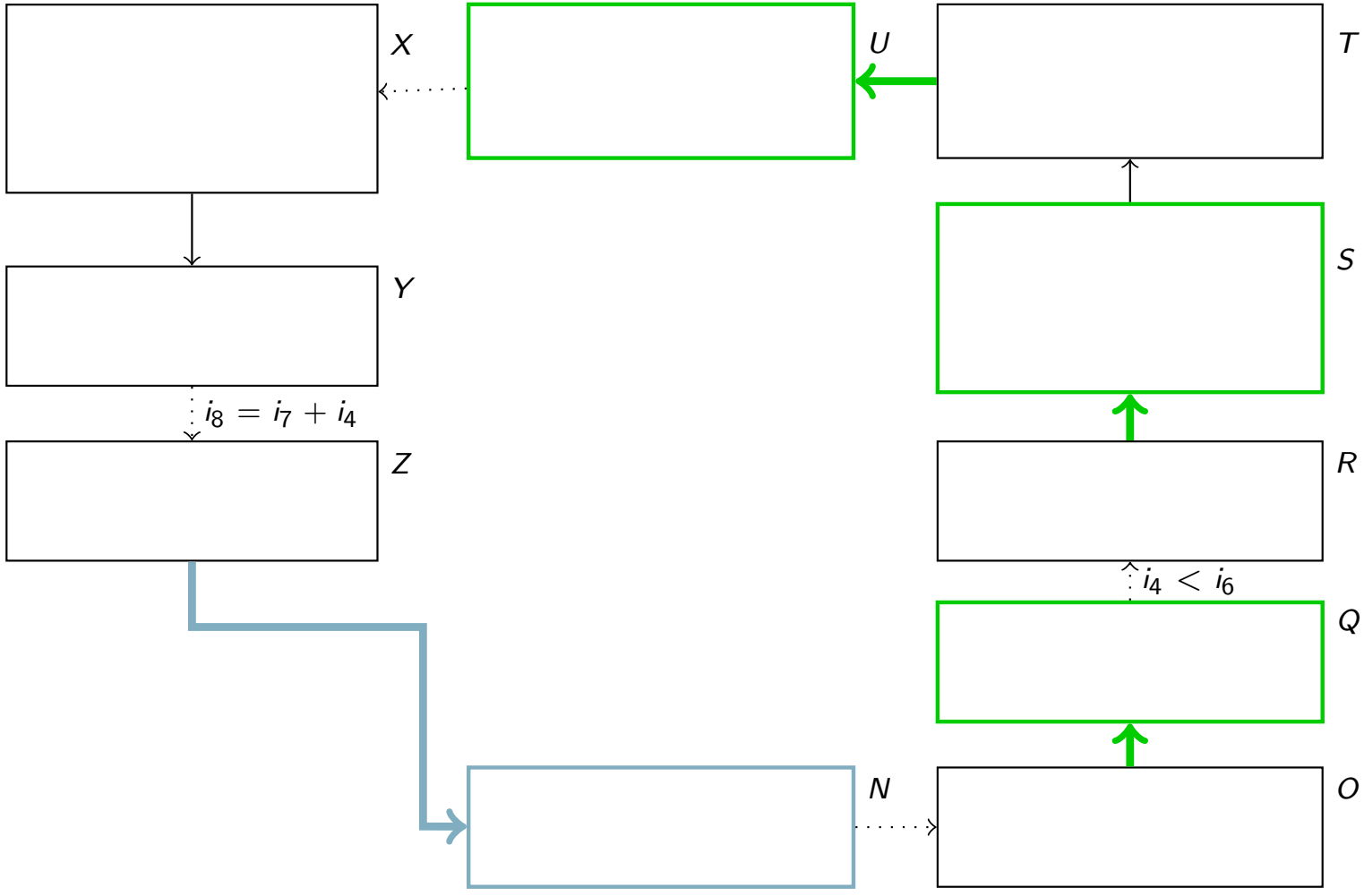
- R refined to S: Re-use current witness - No!
- a_1 too abstract to keep information

$\Rightarrow$ Instantiate array and index by concrete values

- Rest analogously, yields witness



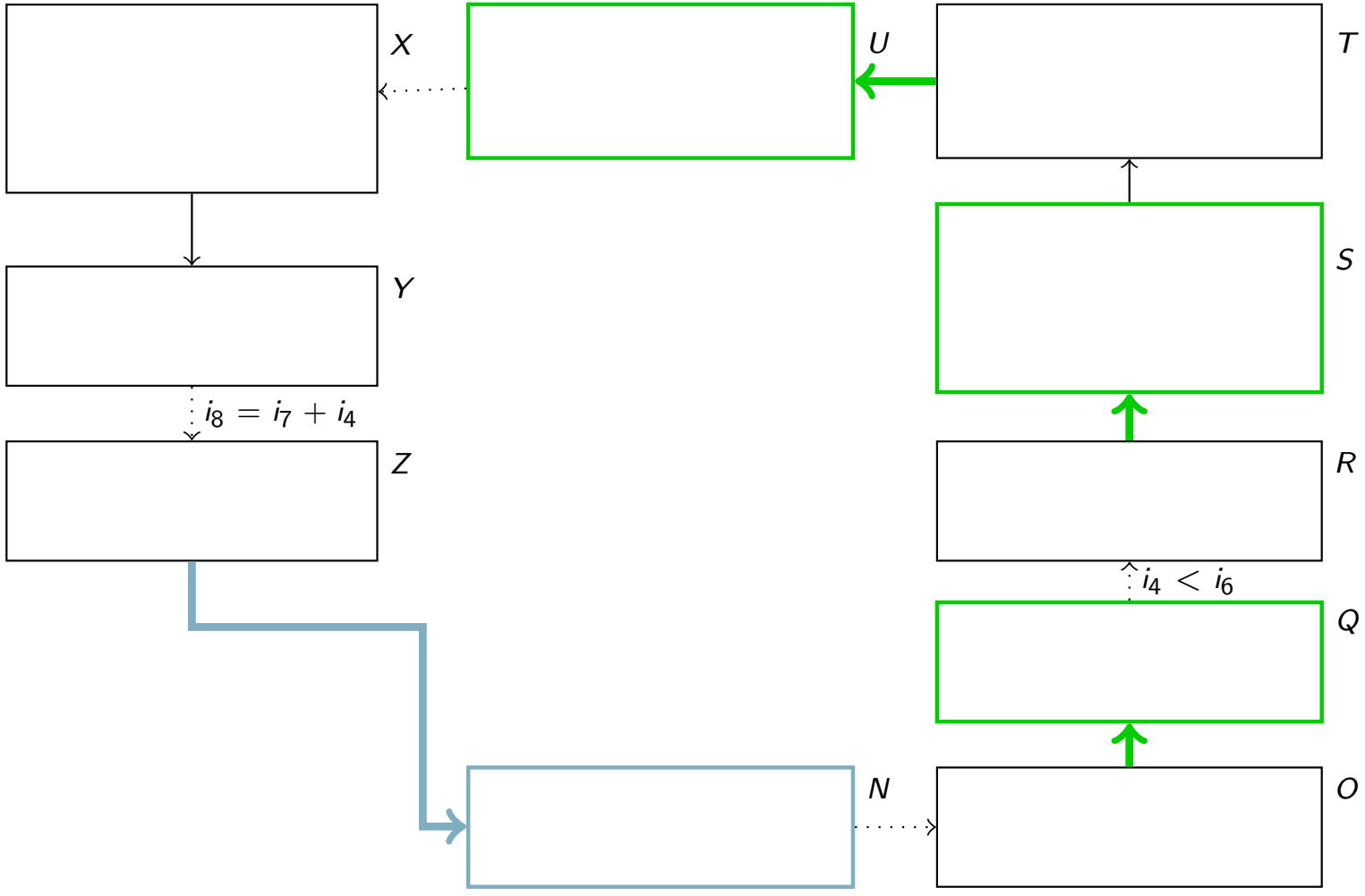
- Goal: Find state that is visited again and again



- Goal: Find state that is visited again and again

$\Rightarrow$ Only consider cycles

- Heuristic: Integers: Find values by SMT
  Objects: Find values by witness generation

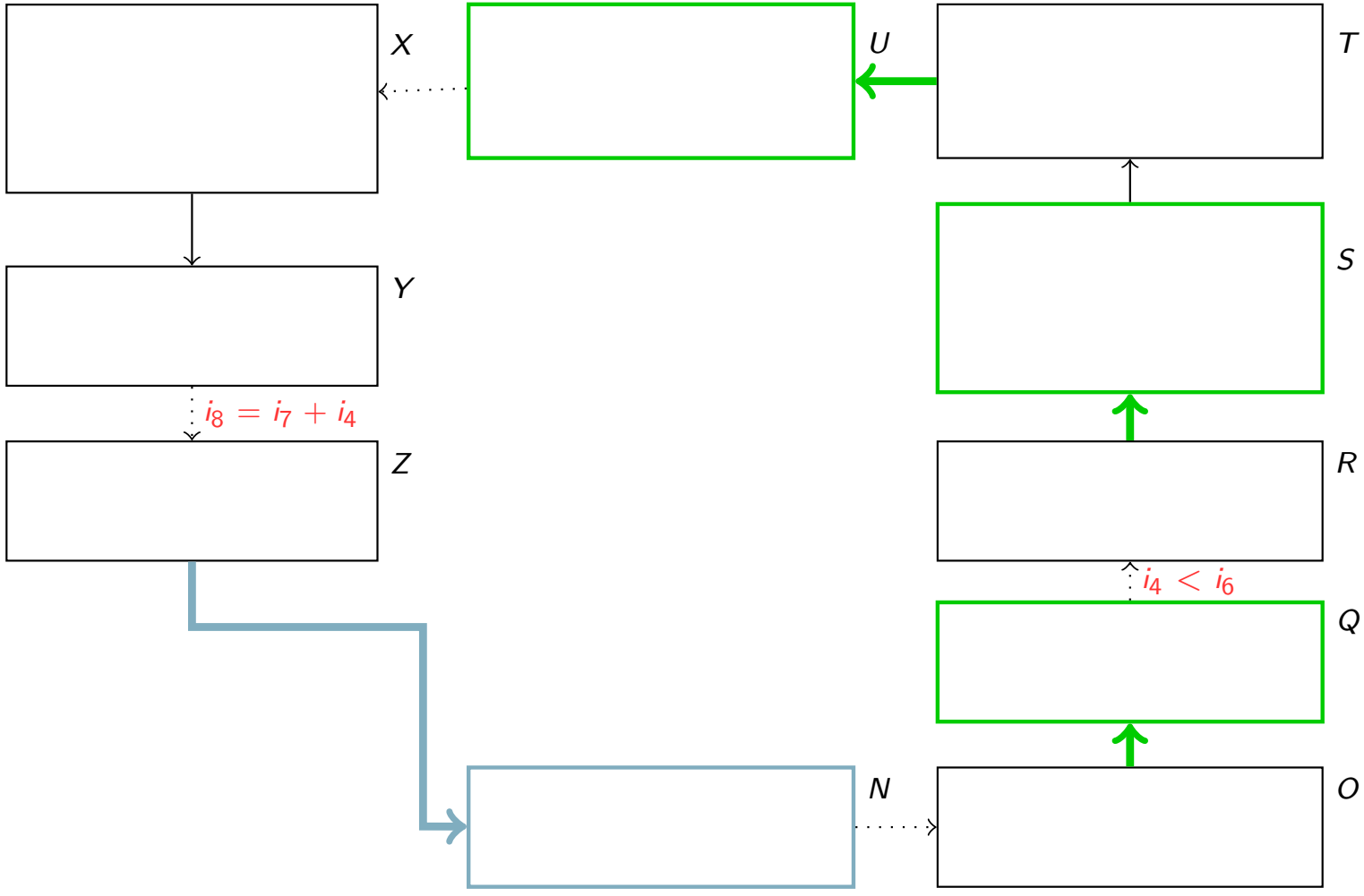


- Goal: Find state that is visited again and again

$\Rightarrow$ Only consider cycles

- Heuristic: Integers: Find values by SMT
  Objects: Find values by witness generation
- Verification: By symbolic evaluation

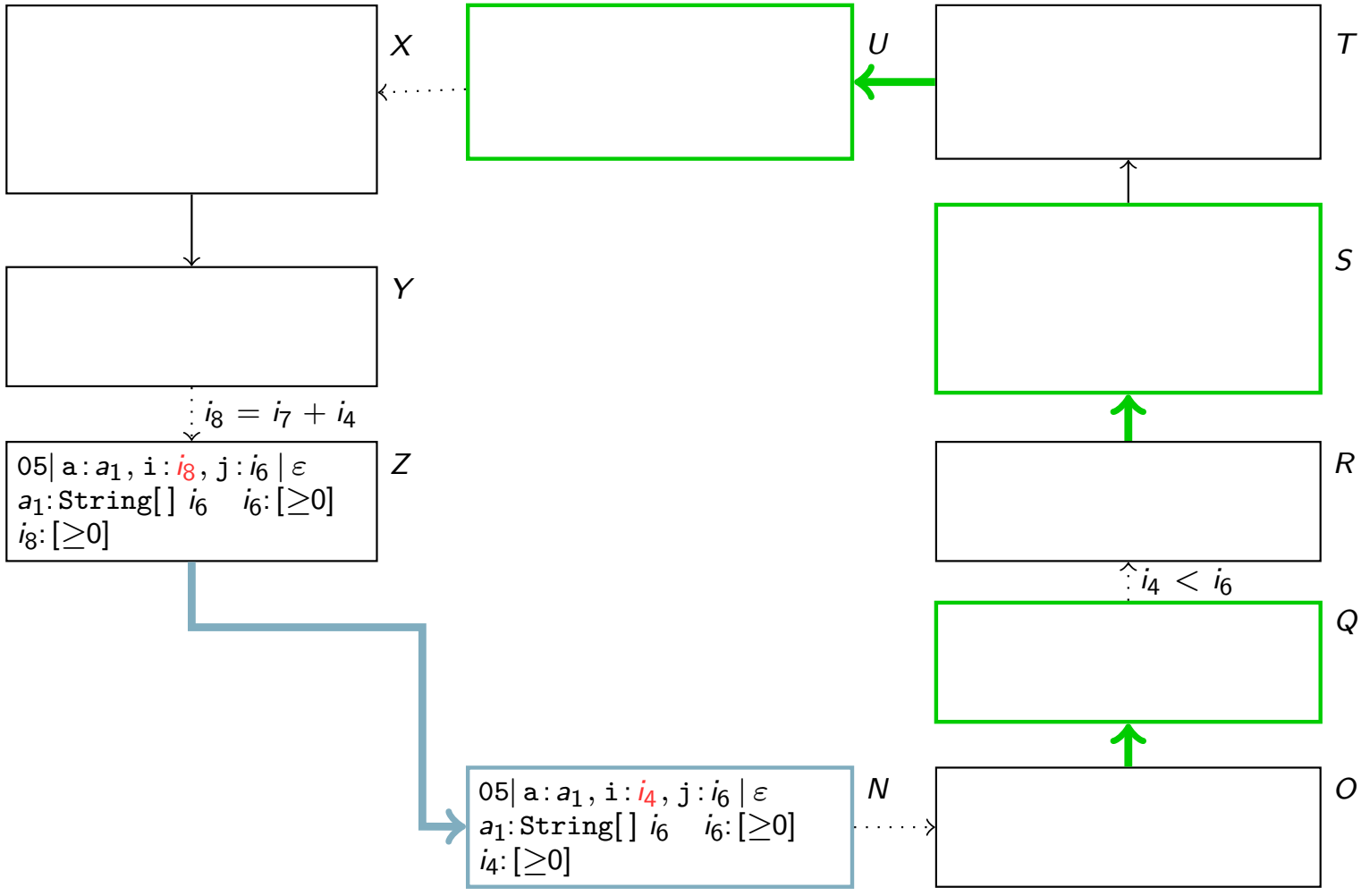
$i_4 < i_6$
$\wedge\ i_8 = i_7 + i_4$



Find values by SMT

- Use conditions on edges

$$
\begin{aligned}
& i_4 < i_6 \\
\wedge\ & i_8 = i_7 + i_4 \\
\wedge\ & i_4 = i_8
\end{aligned}
$$

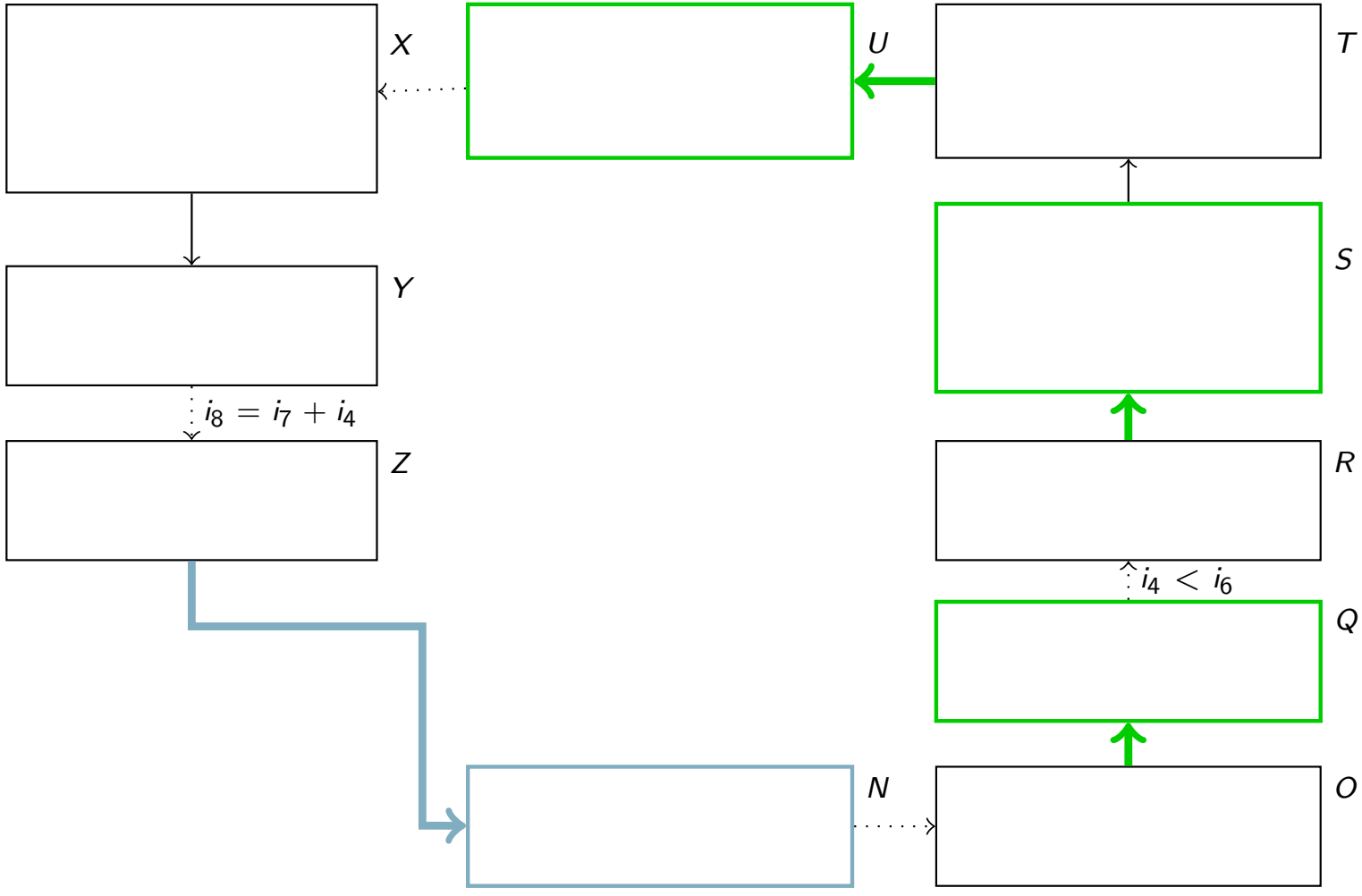


Find values by SMT

- Use conditions on edges
- Refinement, Instance renamings $\Rightarrow$ Equalities

$\Rightarrow$ Last instance edge $\sim$ values stay the same

$$i_4 < i_6 \\ \wedge\ i_8 = i_7 + i_4 \\ \wedge\ i_4 = i_8$$

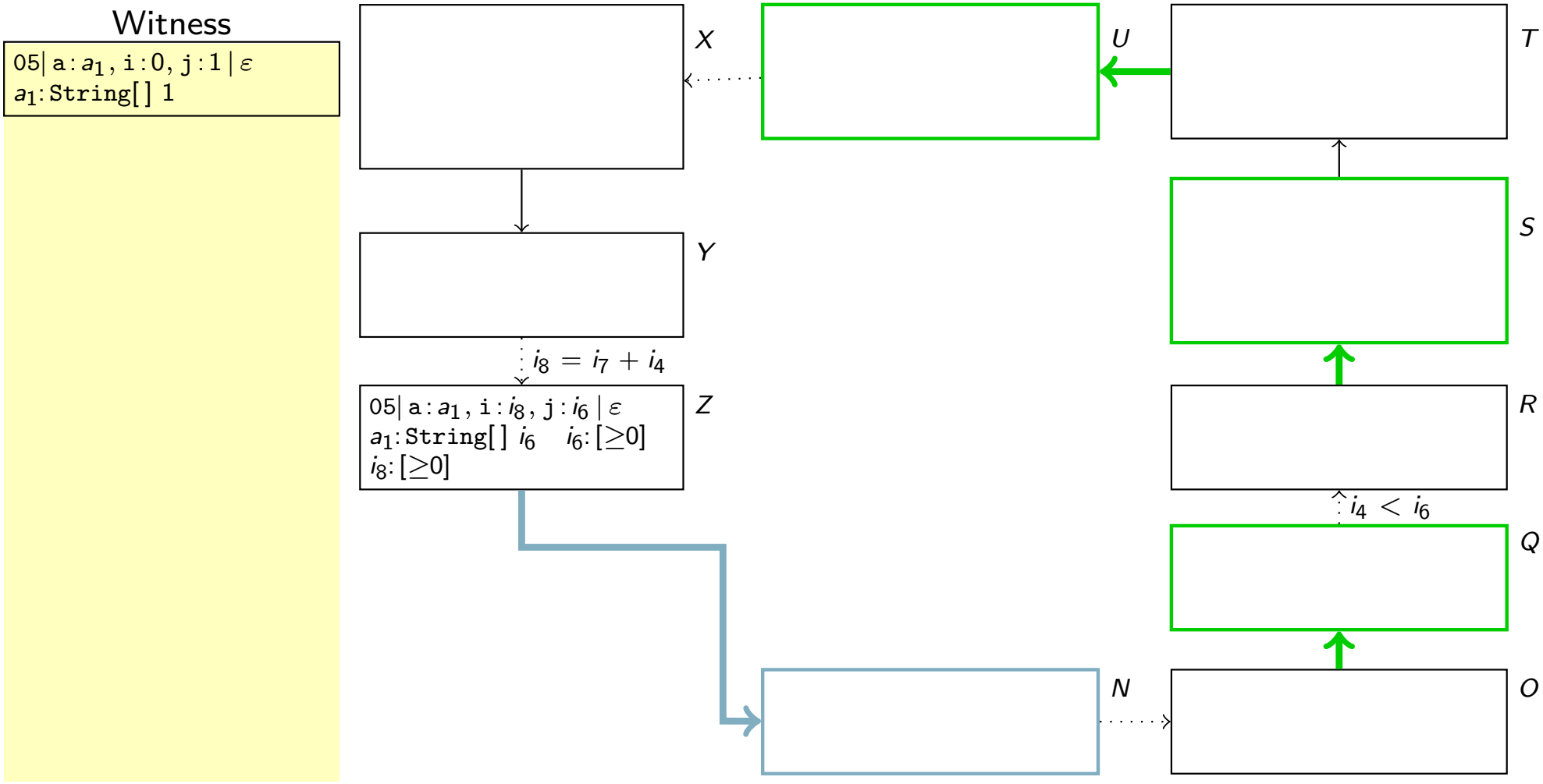


Find values by SMT

- Use conditions on edges
- Refinement, Instance renamings $\Rightarrow$ Equalities

$\Rightarrow$ Last instance edge $\sim$ values stay the same

SMT Solution:
$i_4 = i_7 = i_8 = 0$
$i_6 = 1$

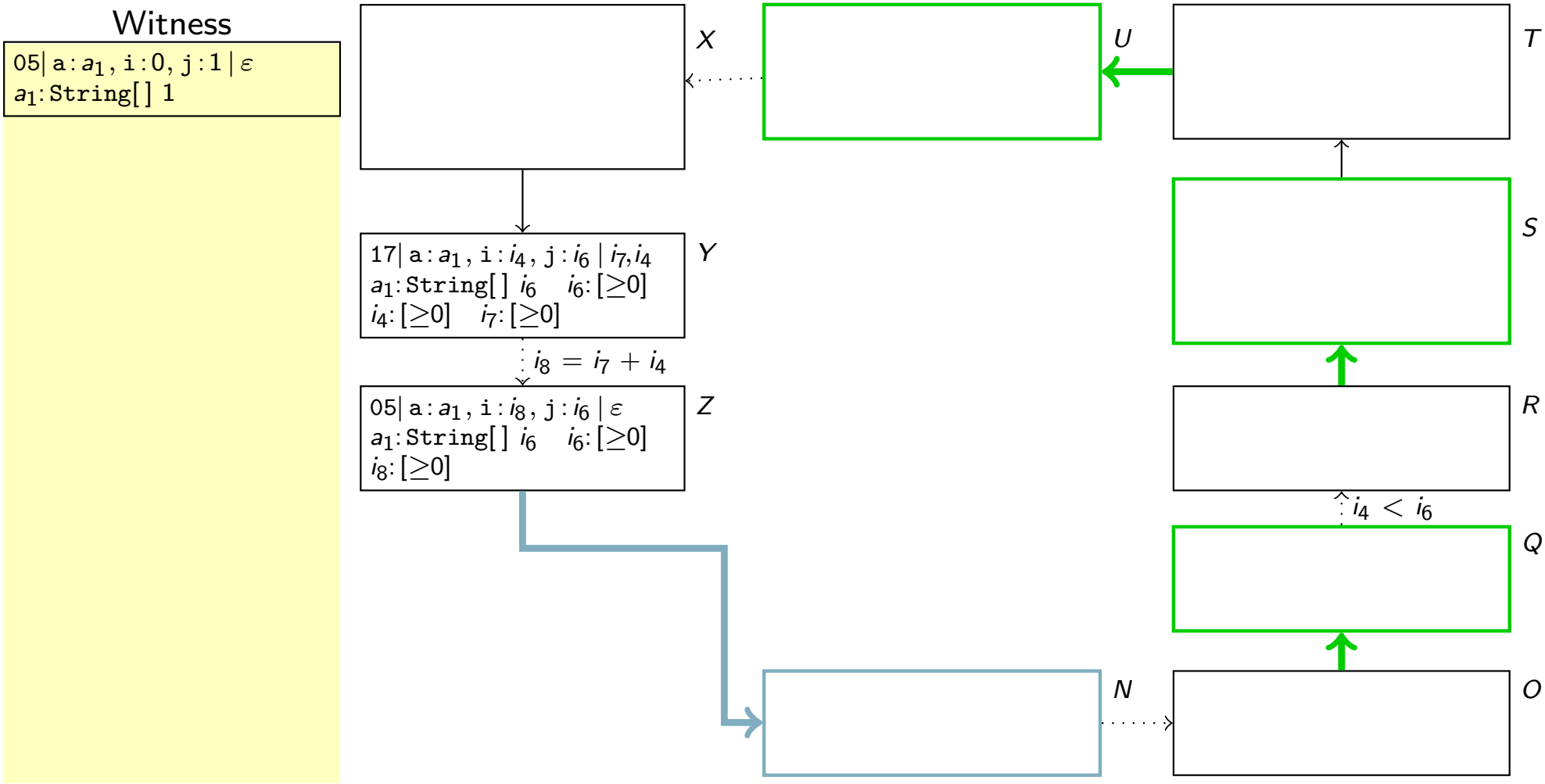


Find values by witness generation

- Instantiate last loop state with integer solution

SMT Solution:

$i_4 = i_7 = i_8 = 0$
$i_6 = 1$

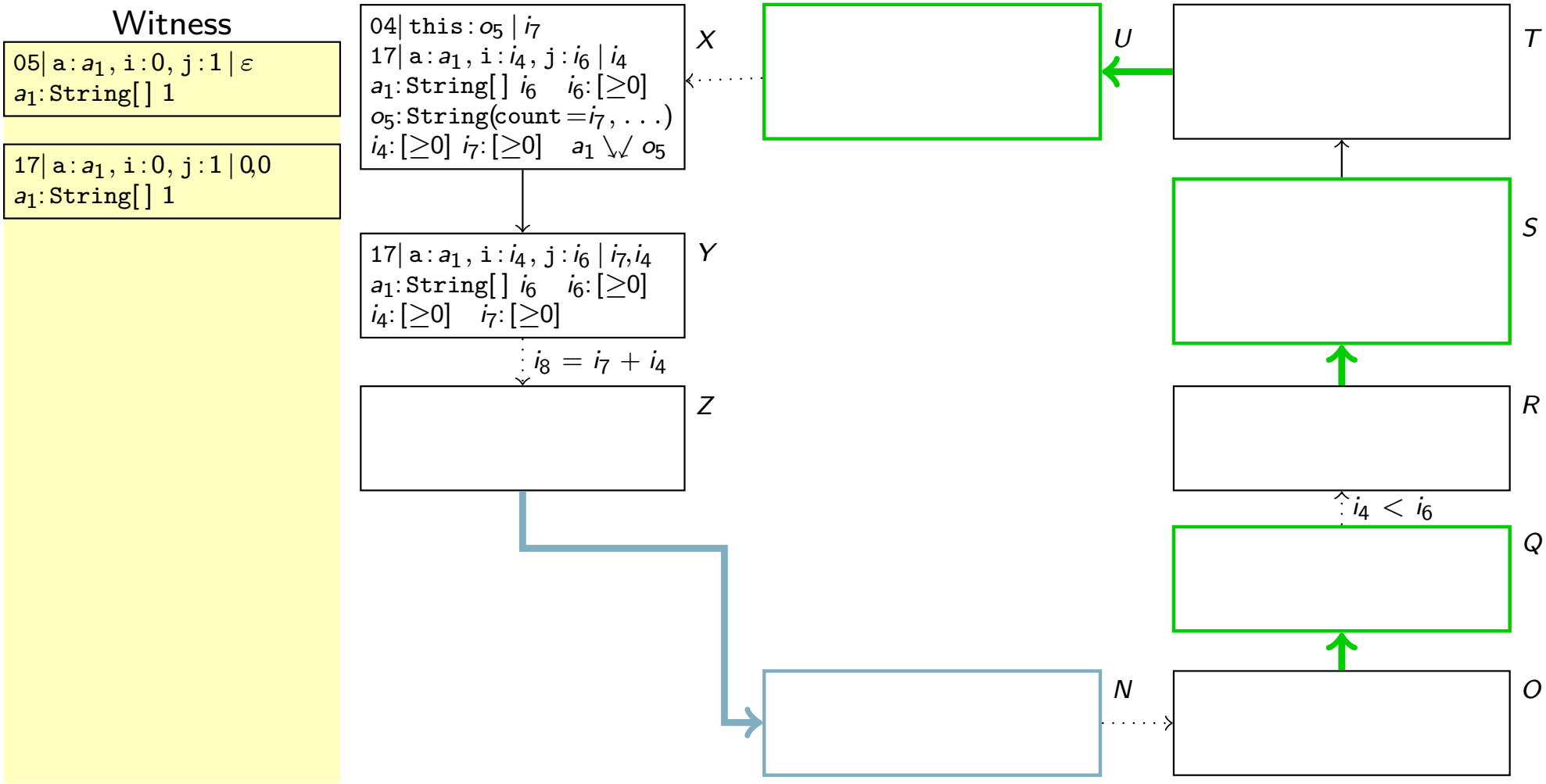


Find values by witness generation

- Instantiate last loop state with integer solution
- Perform witness generation

SMT Solution:

$i_4 = i_7 = i_8 = 0$
$i_6 = 1$



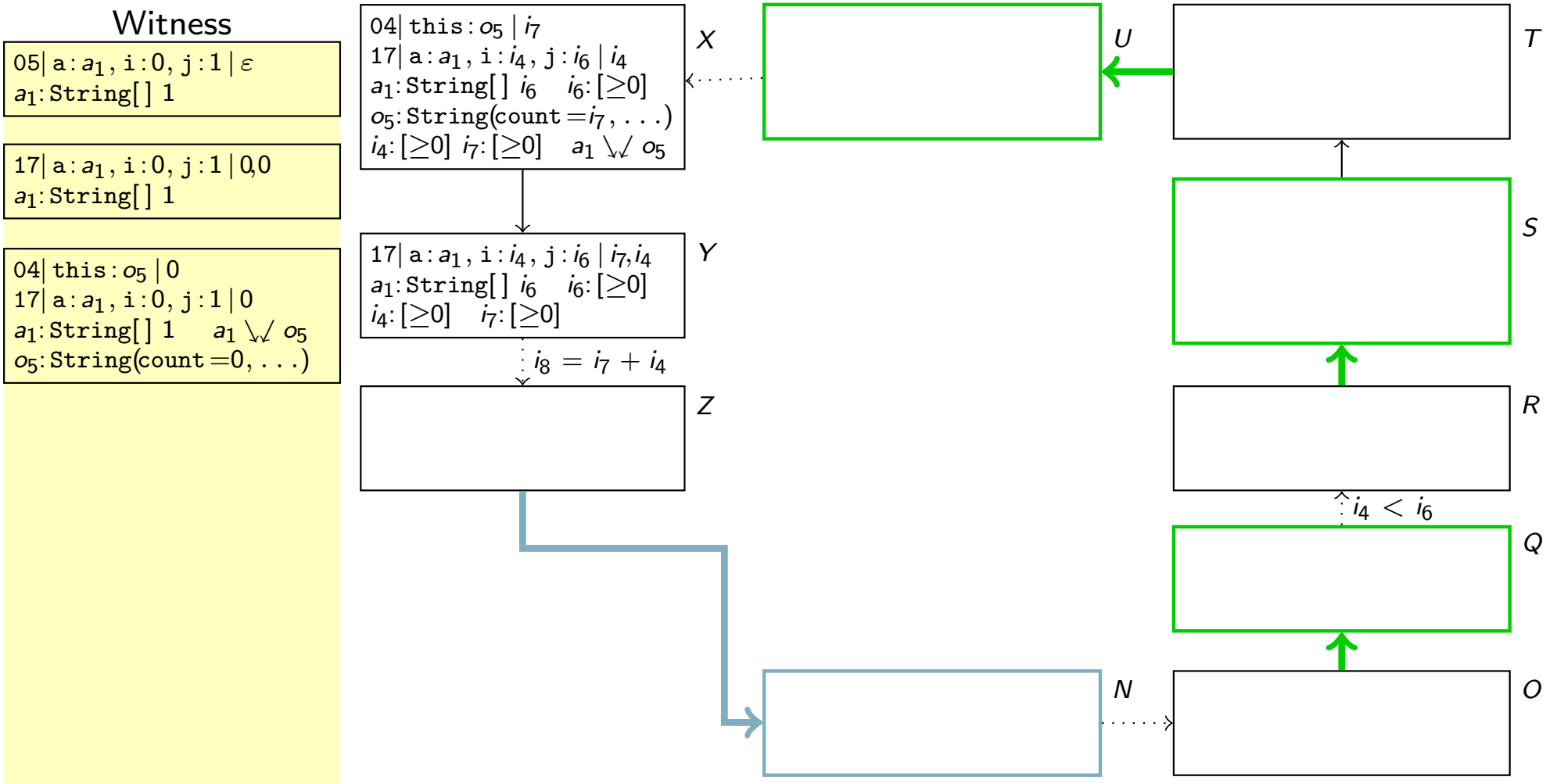
Find values by witness generation

- Instantiate last loop state with integer solution
- Perform witness generation
- Use integer solution if possible

SMT Solution:

$i_4 = i_7 = i_8 = 0$

$i_6 = 1$

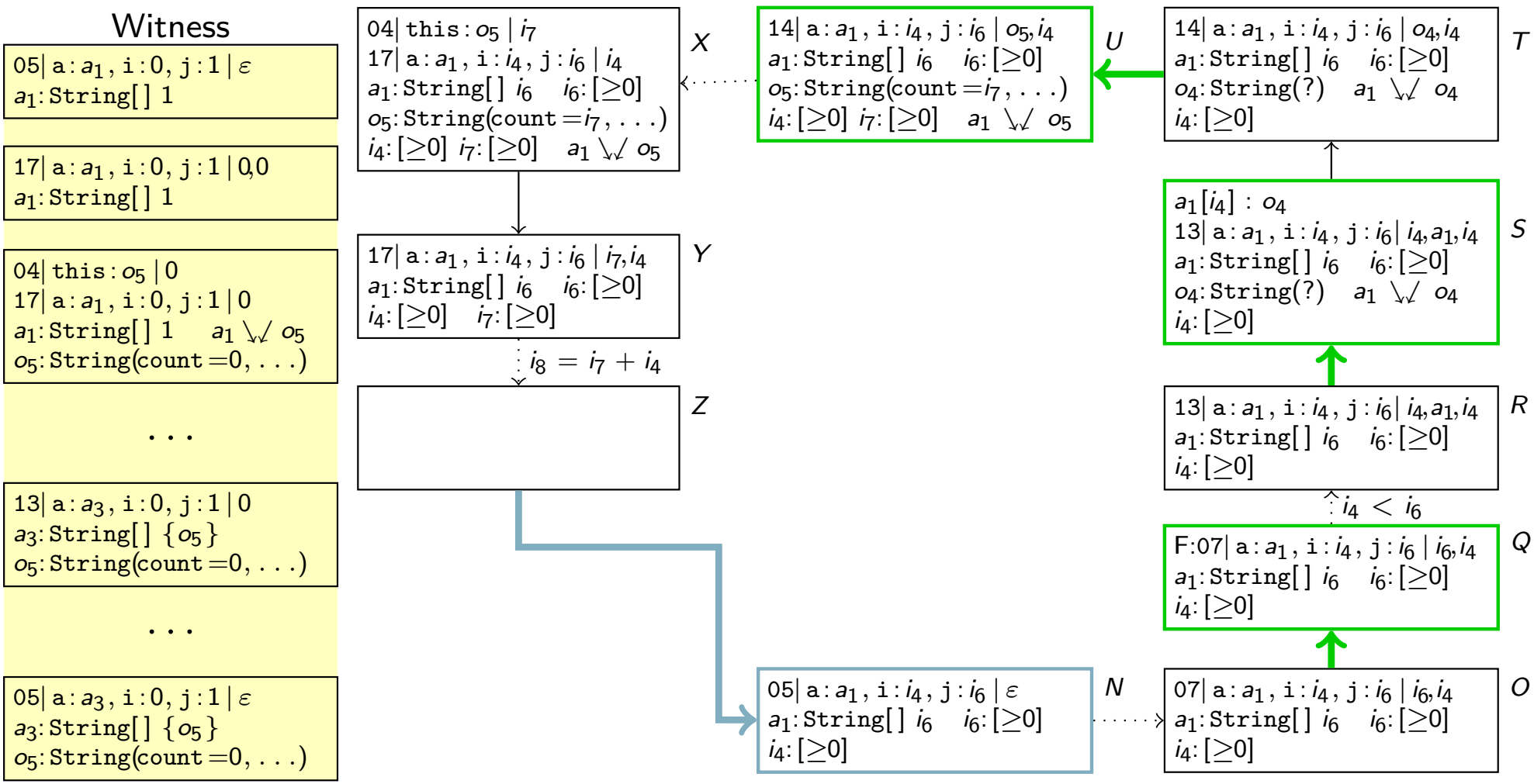


Find values by witness generation

- Instantiate last loop state with integer solution
- Perform witness generation
- Use integer solution if possible

SMT Solution:

$i_4 = i_7 = i_8 = 0$
$i_6 = 1$



Find values by witness generation

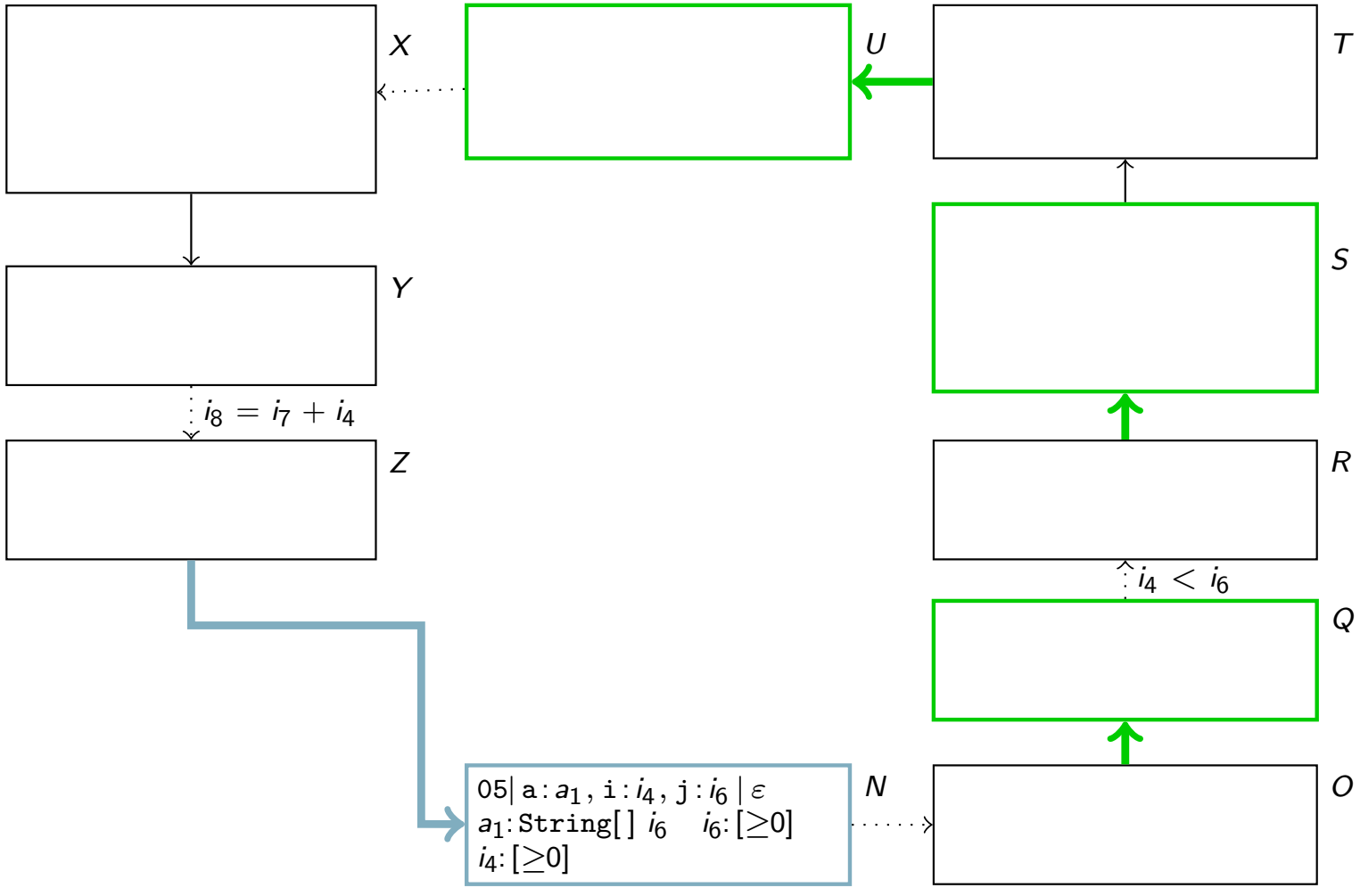
- Instantiate last loop state with integer solution
- Perform witness generation
- Use integer solution if possible
- Continue until state at loop start

SMT Solution:

$i_4 = i_7 = i_8 = 0$

$i_6 = 1$

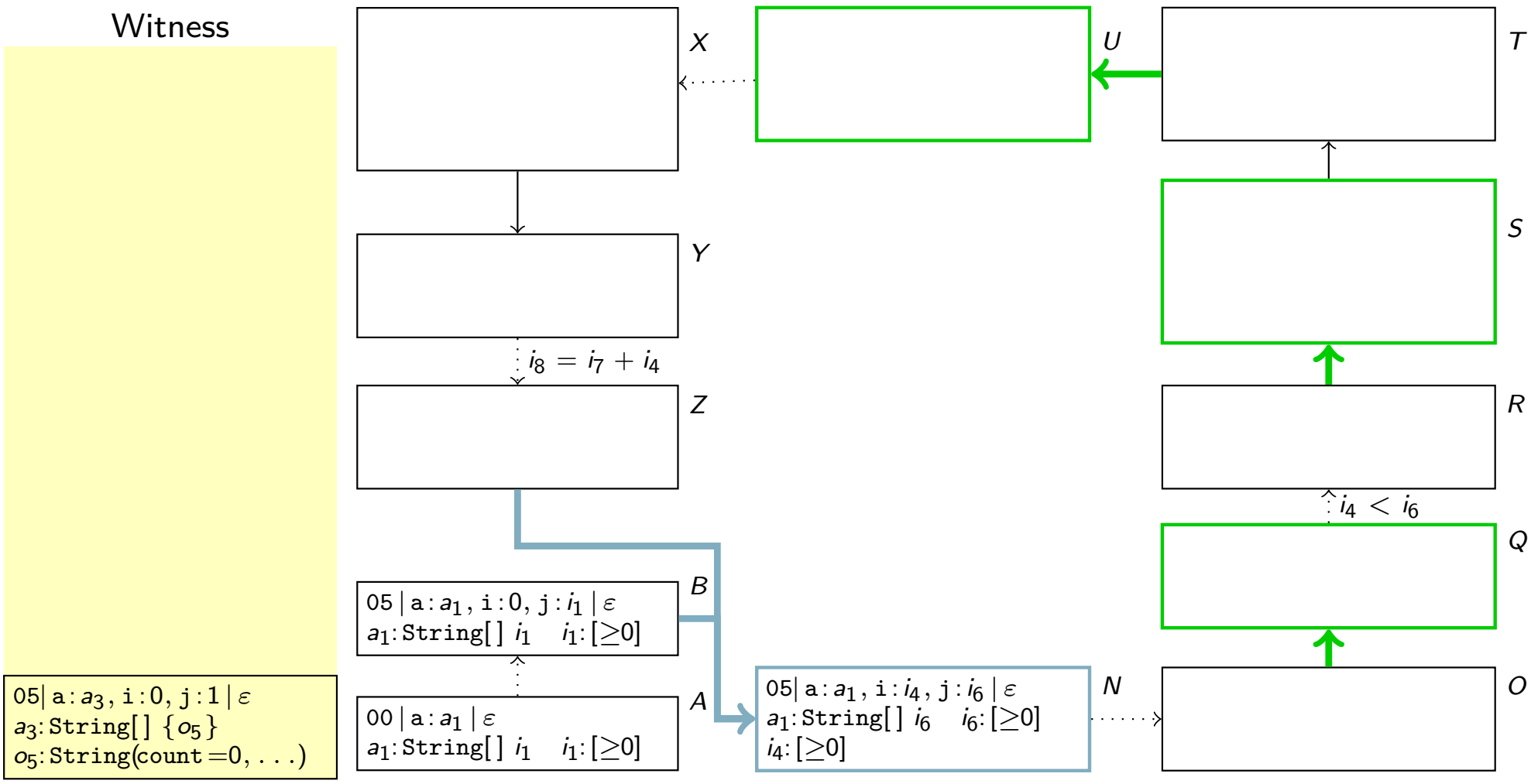
Witness



Verification

- Check that object values stay the same:

$\Rightarrow$ Perform symbolic evaluation.



Verification

- Check that object values stay the same:

$\Rightarrow$ Perform symbolic evaluation.

00| a : a_3 | ε
a_3 : String[ ] {o_5}
o_5 : String(count = 0, . . .)

Witness

- Find start state witness

Other work

Already done:

- Prove correctness
- Non-looping Non-termination
- Termination analysis

Other work

Already done:

- Prove correctness
- Non-looping Non-termination
- Termination analysis

Extensions:

- Use other SMT theories
- Use source code annotations
- ...

Non-termination analysis of JBC

- Implemented in AProVE for full single-threaded Java

Non-termination analysis of JBC

- Implemented in AProVE for full single-threaded Java
- Evaluated on 325 examples:
  - 268 from the *Termination Problem Data Base*
  - 55 from the evaluation of Invel
  - 2 examples from our paper

Non-termination analysis of JBC

- Implemented in AProVE for full single-threaded Java
- Evaluated on 325 examples:
  - 268 from the *Termination Problem Data Base*
  - 55 from the evaluation of Invel
  - 2 examples from our paper

	Invel Ex.					Other Ex.				
	Y	**N**	**F**	**T**	**R**	**Y**	**N**	**F**	**T**	**R**
AProVE-No	1	51	0	3	5	204	30	12	24	11
AProVE	1	0	5	49	54	204	0	27	39	15
Julia	1	0	54	0	2	166	22	82	0	4
Invel	0	42	13	0	?					

Yes, **N**o, **F**ailed proof, **T**imeout, **R**untime (average)

Non-termination analysis of JBC

- Implemented in AProVE for full single-threaded Java
- Evaluated on 325 examples:
  - 268 from the *Termination Problem Data Base*
  - 55 from the evaluation of Invel
  - 2 examples from our paper

	Invel Ex.					Other Ex.				
	Y	**N**	**F**	**T**	**R**	**Y**	**N**	**F**	**T**	**R**
AProVE-No	1	51	0	3	5	204	30	12	24	11
AProVE	1	0	5	49	54	204	0	27	39	15
Julia	1	0	54	0	2	166	22	82	0	4
Invel	0	42	13	0	?					

Yes, **N**o, **F**ailed proof, **T**imeout, **R**untime (average)

http://aprove.informatik.rwth-aachen.de/eval/JBC-Nonterm